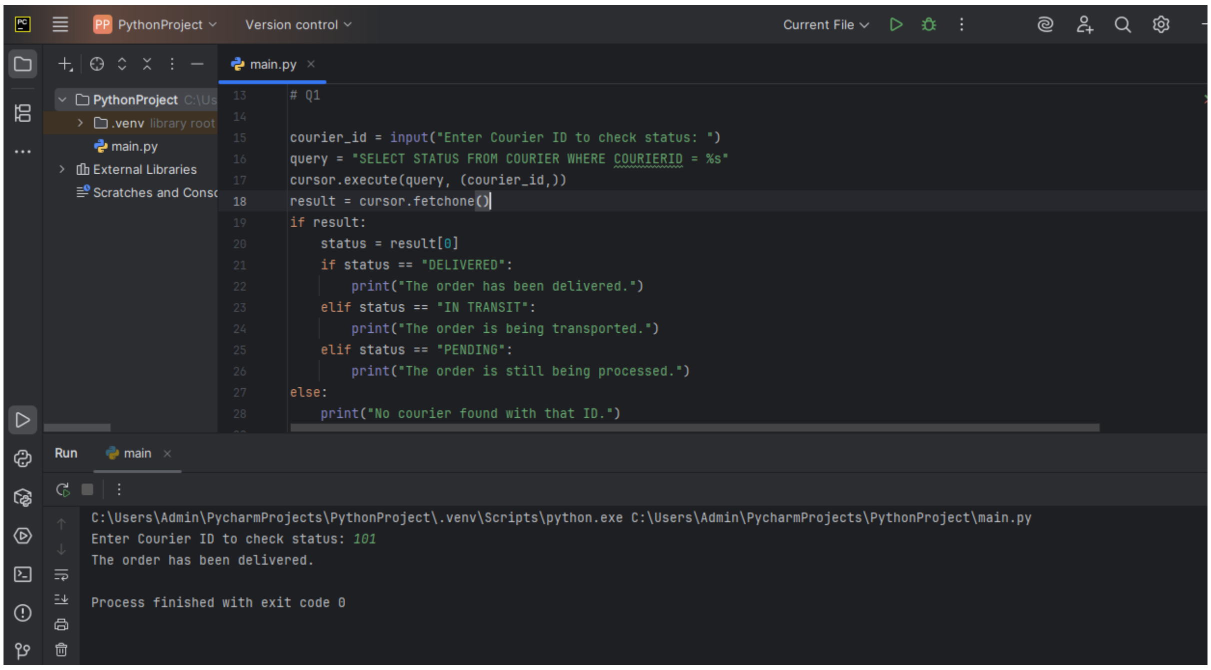
Assignment-Courier Management System

-Elakkiya M

Coding

Task 1: Control Flow Statements

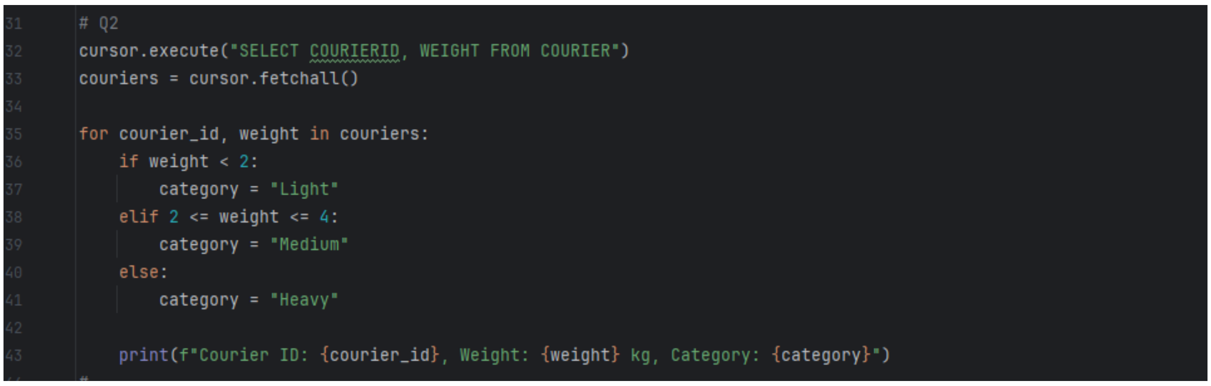
1. Write a program that checks whether a given order is delivered or not based on its status (e.g., "Processing," "Delivered," "Cancelled"). Use if-else statements for this.



```python
# Q1

courier_id = input("Enter Courier ID to check status: ")
query = "SELECT STATUS FROM COURIER WHERE COURIERID = %s"
cursor.execute(query, (courier_id,))
result = cursor.fetchone()
if result:
    status = result[0]
    if status == "DELIVERED":
        print("The order has been delivered.")
    elif status == "IN TRANSIT":
        print("The order is being transported.")
    elif status == "PENDING":
        print("The order is still being processed.")
else:
    print("No courier found with that ID.")
```

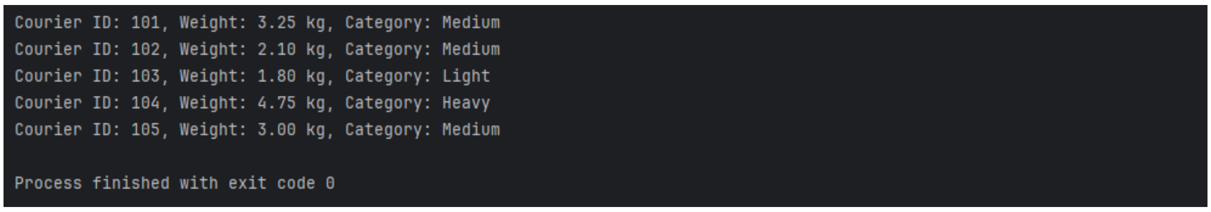
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter Courier ID to check status: 101
The order has been delivered.

Process finished with exit code 0
```

2. Implement a switch-case statement to categorize parcels based on their weight into "Light," "Medium," or "Heavy."
   Program:



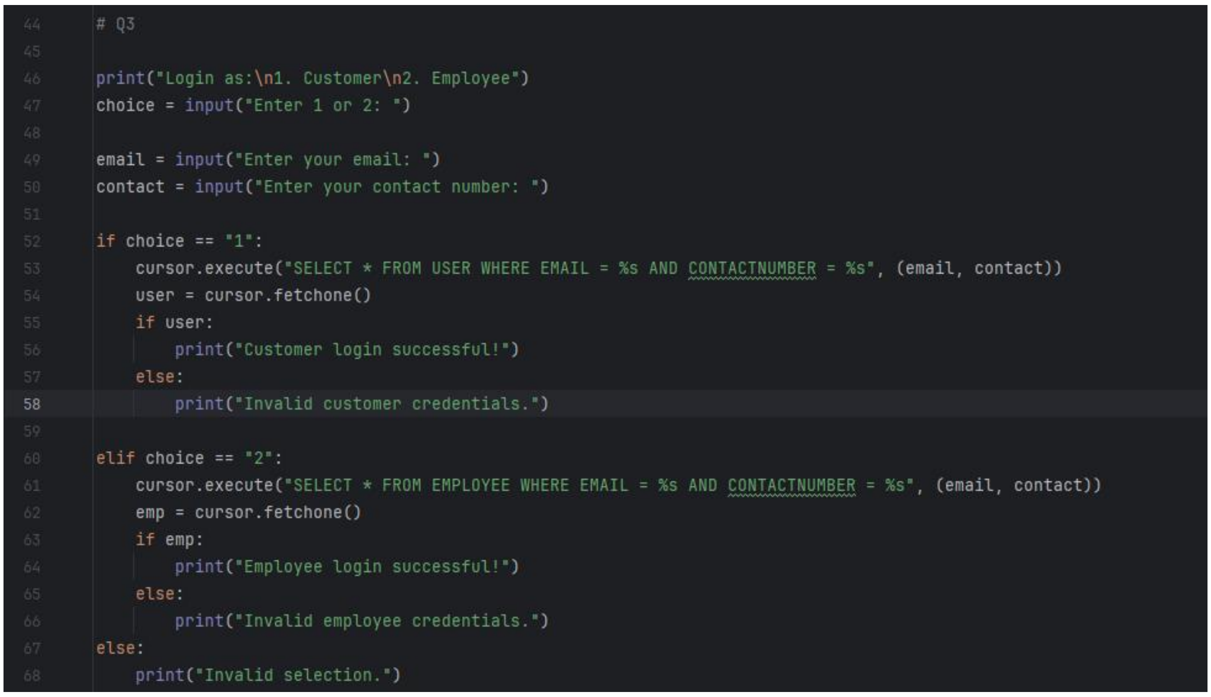
```python
# Q2
cursor.execute("SELECT COURIERID, WEIGHT FROM COURIER")
couriers = cursor.fetchall()

for courier_id, weight in couriers:
    if weight < 2:
        category = "Light"
    elif 2 <= weight <= 4:
        category = "Medium"
    else:
        category = "Heavy"

    print(f"Courier ID: {courier_id}, Weight: {weight} kg, Category: {category}")
```

Output:



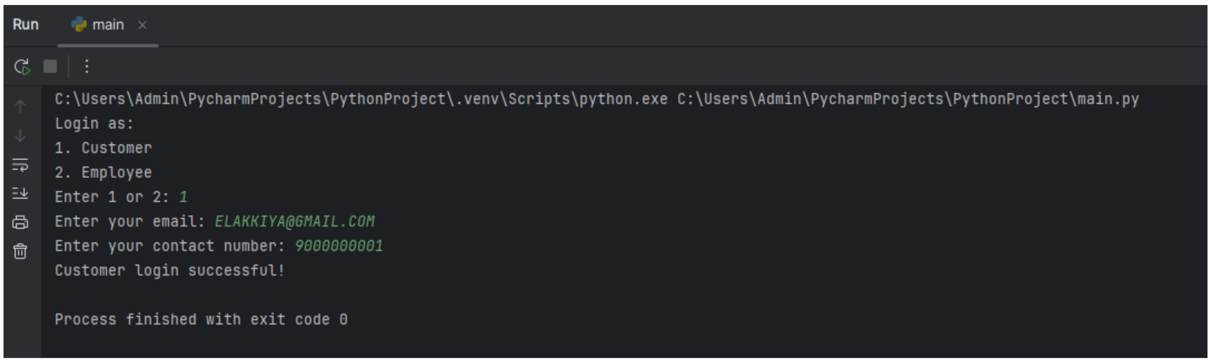
```
Courier ID: 101, Weight: 3.25 kg, Category: Medium
Courier ID: 102, Weight: 2.10 kg, Category: Medium
Courier ID: 103, Weight: 1.80 kg, Category: Light
Courier ID: 104, Weight: 4.75 kg, Category: Heavy
Courier ID: 105, Weight: 3.00 kg, Category: Medium

Process finished with exit code 0
```

3. Implement User Authentication 1. Create a login system for employees and customers using Java control flow statements.



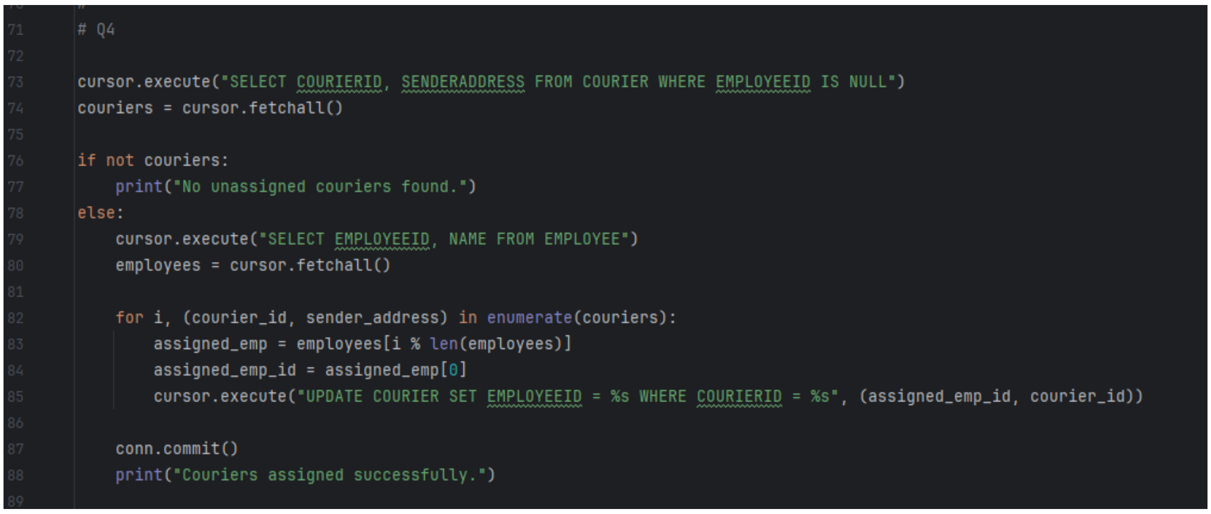
```python
# Q3

print("Login as:\n1. Customer\n2. Employee")
choice = input("Enter 1 or 2: ")

email = input("Enter your email: ")
contact = input("Enter your contact number: ")

if choice == "1":
    cursor.execute("SELECT * FROM USER WHERE EMAIL = %s AND CONTACTNUMBER = %s", (email, contact))
    user = cursor.fetchone()
    if user:
        print("Customer login successful!")
    else:
        print("Invalid customer credentials.")

elif choice == "2":
    cursor.execute("SELECT * FROM EMPLOYEE WHERE EMAIL = %s AND CONTACTNUMBER = %s", (email, contact))
    emp = cursor.fetchone()
    if emp:
        print("Employee login successful!")
    else:
        print("Invalid employee credentials.")
else:
    print("Invalid selection.")
```

Output:



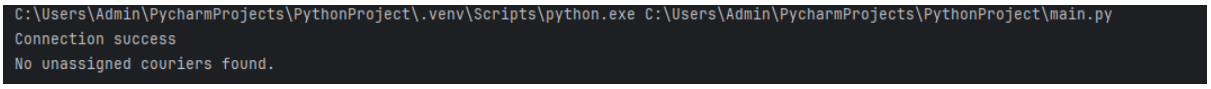
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Login as:
1. Customer
2. Employee
Enter 1 or 2: 1
Enter your email: ELAKKIYA@GMAIL.COM
Enter your contact number: 9000000001
Customer login successful!

Process finished with exit code 0
```

4. Implement Courier Assignment Logic 1. Develop a mechanism to assign couriers to shipments based on predefined criteria (e.g., proximity, load capacity) using loops.



```python
# Q4

cursor.execute("SELECT COURIERID, SENDERADDRESS FROM COURIER WHERE EMPLOYEEID IS NULL")
couriers = cursor.fetchall()

if not couriers:
    print("No unassigned couriers found.")
else:
    cursor.execute("SELECT EMPLOYEEID, NAME FROM EMPLOYEE")
    employees = cursor.fetchall()

    for i, (courier_id, sender_address) in enumerate(couriers):
        assigned_emp = employees[i % len(employees)]
        assigned_emp_id = assigned_emp[0]
        cursor.execute("UPDATE COURIER SET EMPLOYEEID = %s WHERE COURIERID = %s", (assigned_emp_id, courier_id))

    conn.commit()
    print("Couriers assigned successfully.")
```

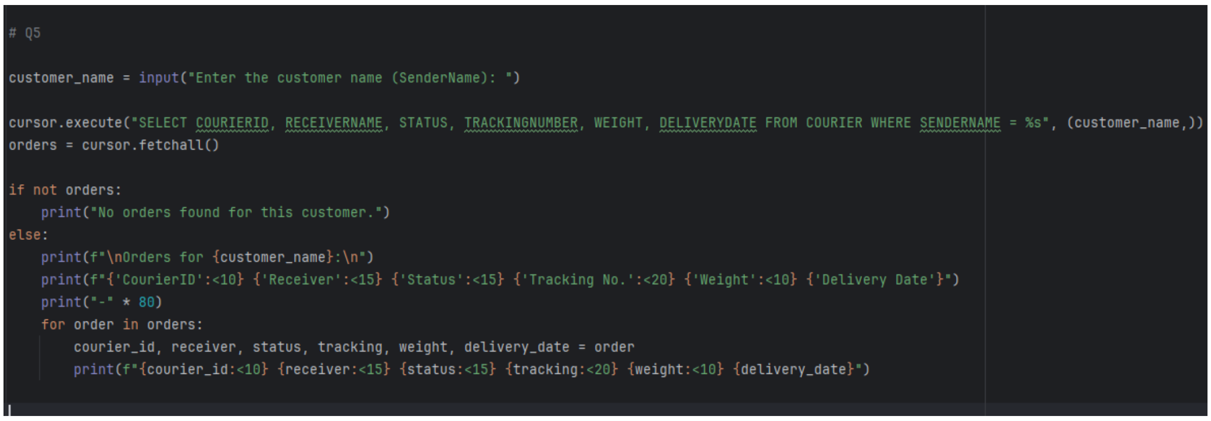
Output:



```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Connection success
No unassigned couriers found.
```

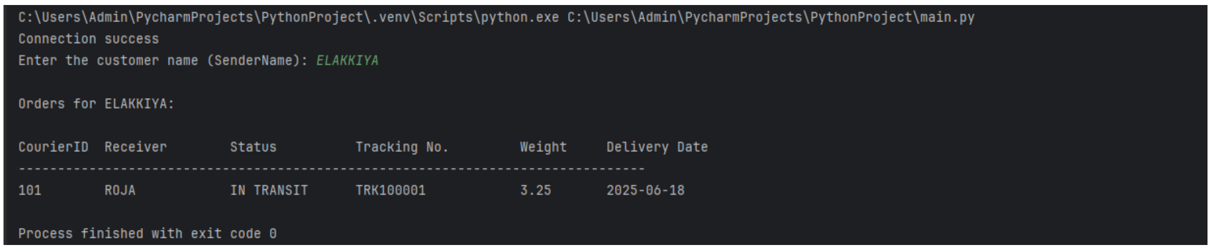
Task 2: Loops and Iteration

5. Write a python program that uses a for loop to display all the orders for a specific customer.



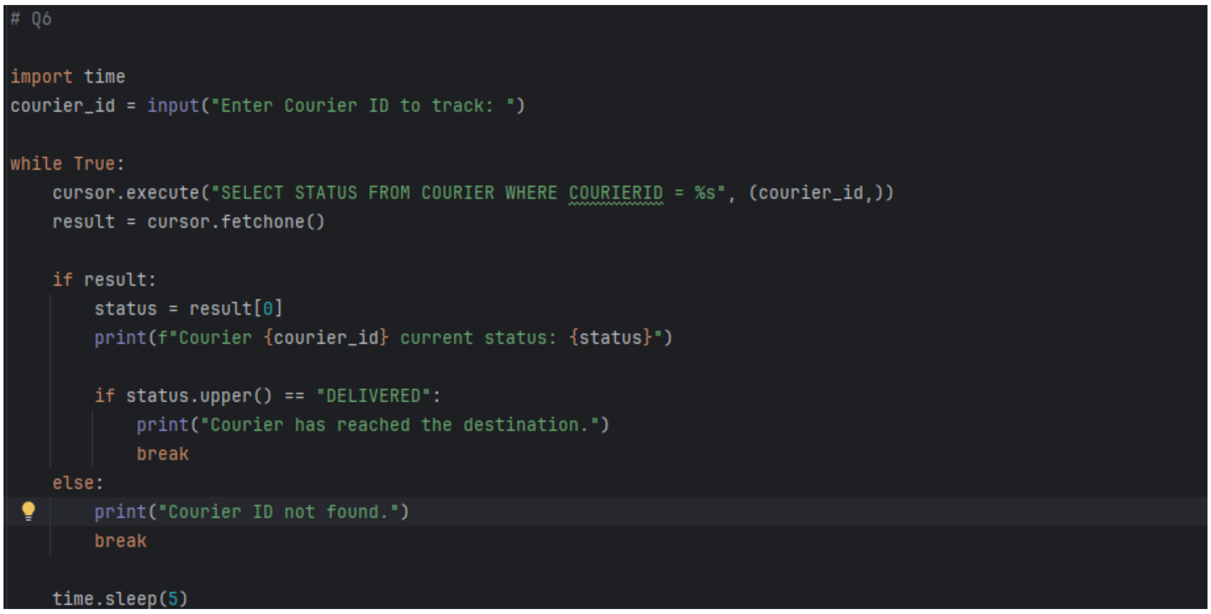
```python
# Q5

customer_name = input("Enter the customer name (SenderName): ")

cursor.execute("SELECT COURIERID, RECEIVERNAME, STATUS, TRACKINGNUMBER, WEIGHT, DELIVERYDATE FROM COURIER WHERE SENDERNAME = %s", (customer_name,))
orders = cursor.fetchall()

if not orders:
    print("No orders found for this customer.")
else:
    print(f"\nOrders for {customer_name}:\n")
    print(f"{'CourierID':<10} {'Receiver':<15} {'Status':<15} {'Tracking No.':<20} {'Weight':<10} {'Delivery Date'}")
    print("-" * 80)
    for order in orders:
        courier_id, receiver, status, tracking, weight, delivery_date = order
        print(f"{courier_id:<10} {receiver:<15} {status:<15} {tracking:<20} {weight:<10} {delivery_date}")
```

Output:



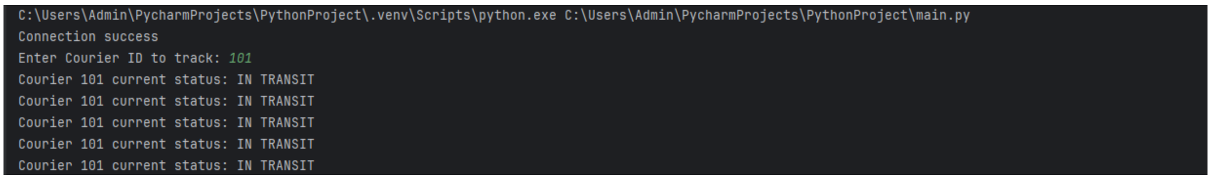
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Connection success
Enter the customer name (SenderName): ELAKKIYA

Orders for ELAKKIYA:

CourierID  Receiver        Status          Tracking No.         Weight     Delivery Date
--------------------------------------------------------------------------------
101        ROJA            IN TRANSIT      TRK100001            3.25       2025-06-18

Process finished with exit code 0
```

6. Implement a while loop to track the real-time location of a courier until it reaches its destination.



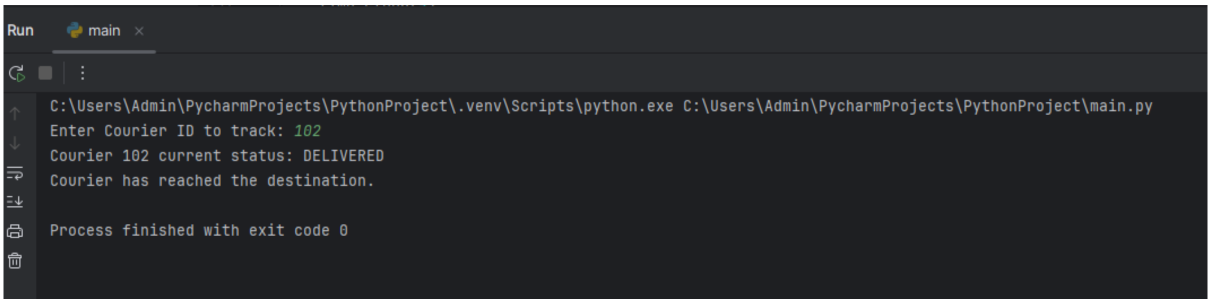
```
# Q6

import time
courier_id = input("Enter Courier ID to track: ")

while True:
    cursor.execute("SELECT STATUS FROM COURIER WHERE COURIERID = %s", (courier_id,))
    result = cursor.fetchone()

    if result:
        status = result[0]
        print(f"Courier {courier_id} current status: {status}")

        if status.upper() == "DELIVERED":
            print("Courier has reached the destination.")
            break
    else:
        print("Courier ID not found.")
        break

    time.sleep(5)
```

Output:



```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Connection success
Enter Courier ID to track: 101
Courier 101 current status: IN TRANSIT
Courier 101 current status: IN TRANSIT
Courier 101 current status: IN TRANSIT
Courier 101 current status: IN TRANSIT
Courier 101 current status: IN TRANSIT
```

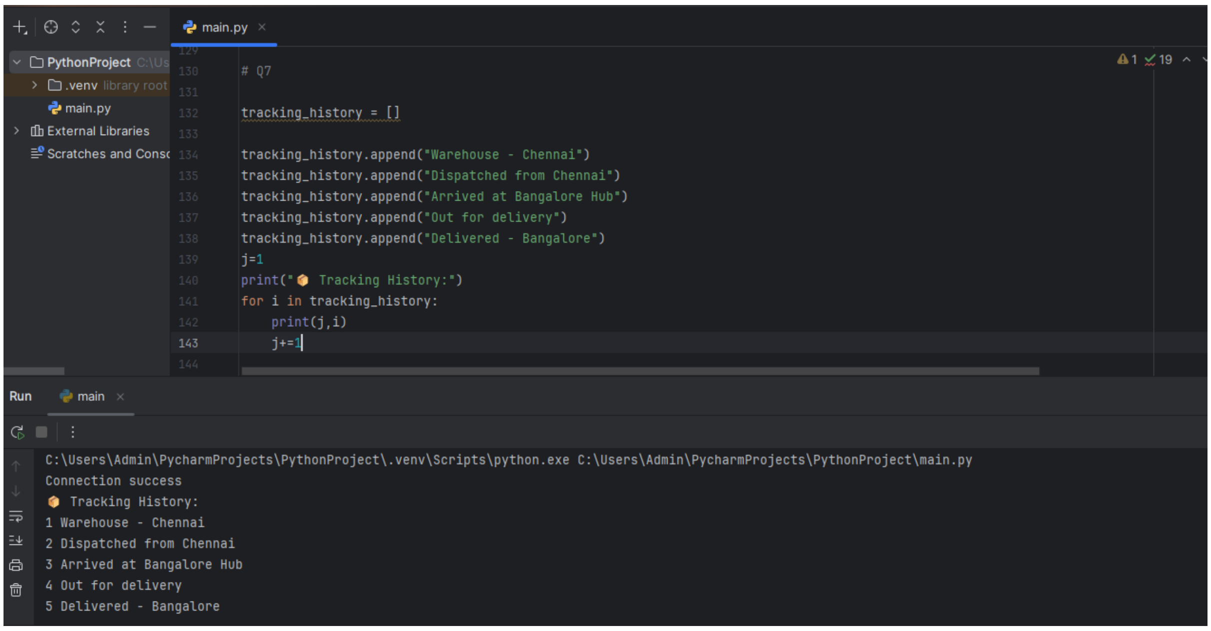
The loop ends only when the status of the courier ID is updated as delivered.



```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter Courier ID to track: 102
Courier 102 current status: DELIVERED
Courier has reached the destination.

Process finished with exit code 0
```

Task 3: Arrays and Data Structures

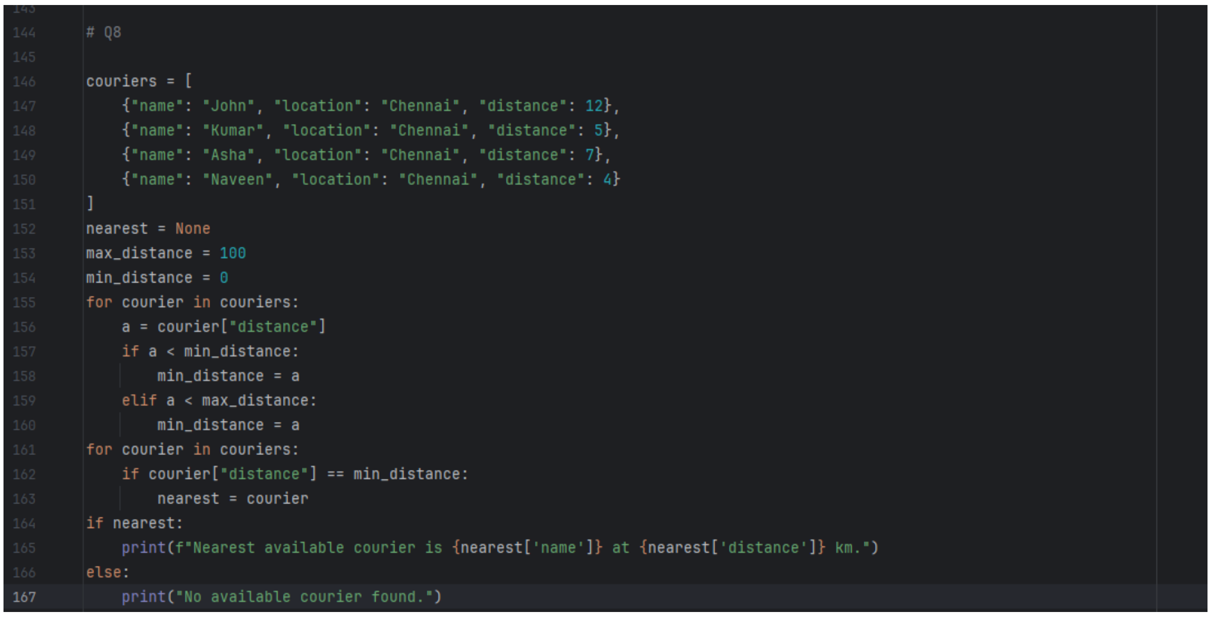
7. Create an array to store the tracking history of a parcel, where each entry represents a location update.



```python
# Q7

tracking_history = []

tracking_history.append("Warehouse - Chennai")
tracking_history.append("Dispatched from Chennai")
tracking_history.append("Arrived at Bangalore Hub")
tracking_history.append("Out for delivery")
tracking_history.append("Delivered - Bangalore")
j=1
print("📦 Tracking History:")
for i in tracking_history:
    print(j,i)
    j+=1
```

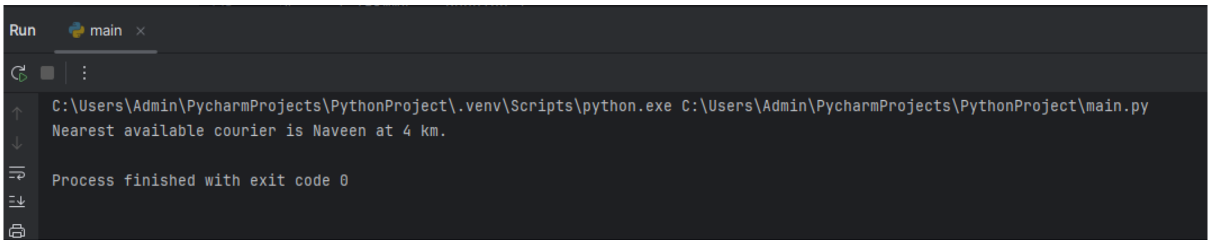
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Connection success
📦 Tracking History:
1 Warehouse - Chennai
2 Dispatched from Chennai
3 Arrived at Bangalore Hub
4 Out for delivery
5 Delivered - Bangalore
```

8. Implement a method to find the nearest available courier for a new order using an array of couriers.



```python
# Q8

couriers = [
    {"name": "John", "location": "Chennai", "distance": 12},
    {"name": "Kumar", "location": "Chennai", "distance": 5},
    {"name": "Asha", "location": "Chennai", "distance": 7},
    {"name": "Naveen", "location": "Chennai", "distance": 4}
]
nearest = None
max_distance = 100
min_distance = 0
for courier in couriers:
    a = courier["distance"]
    if a < min_distance:
        min_distance = a
    elif a < max_distance:
        min_distance = a
for courier in couriers:
    if courier["distance"] == min_distance:
        nearest = courier
if nearest:
    print(f"Nearest available courier is {nearest['name']} at {nearest['distance']} km.")
else:
    print("No available courier found.")
```

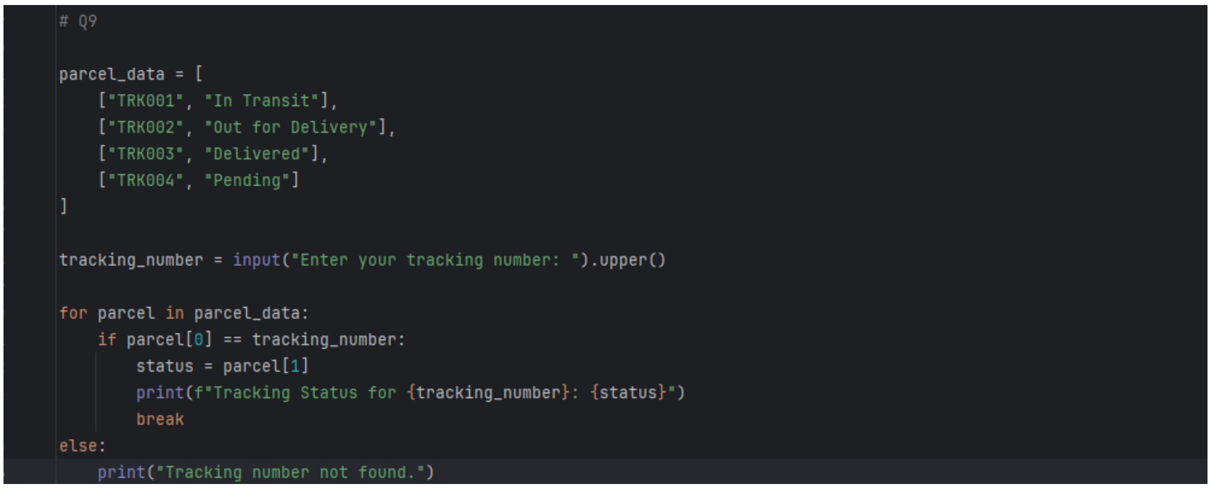
Output:



```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Nearest available courier is Naveen at 4 km.

Process finished with exit code 0
```

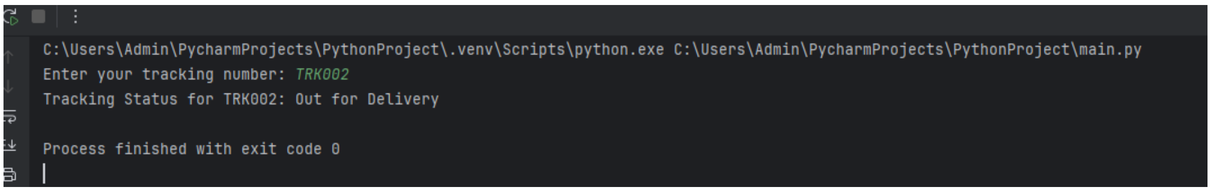
Task 4: Strings,2d Arrays, user defined functions,Hashmap

9. Parcel Tracking: Create a program that allows users to input a parcel tracking number.Store the tracking number and Status in 2d String Array. Initialize the array with values. Then, simulate the tracking process by displaying messages like "Parcel in transit," "Parcel out for delivery," or "Parcel delivered" based on the tracking number's status.



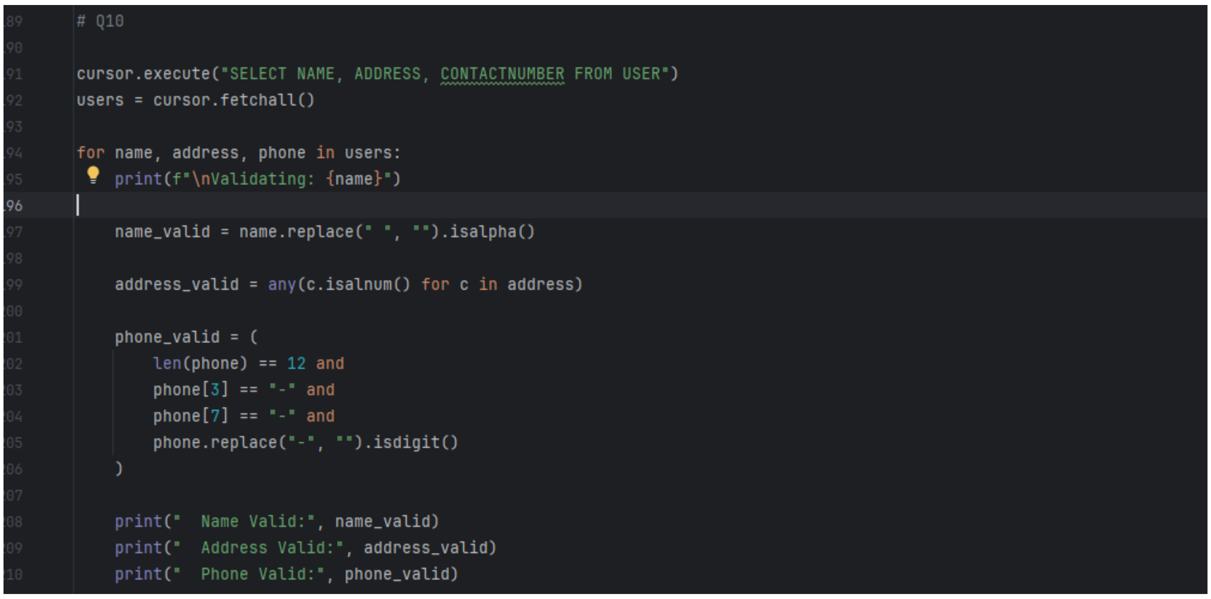
```python
# Q9

parcel_data = [
    ["TRK001", "In Transit"],
    ["TRK002", "Out for Delivery"],
    ["TRK003", "Delivered"],
    ["TRK004", "Pending"]
]

tracking_number = input("Enter your tracking number: ").upper()

for parcel in parcel_data:
    if parcel[0] == tracking_number:
        status = parcel[1]
        print(f"Tracking Status for {tracking_number}: {status}")
        break
else:
    print("Tracking number not found.")
```

Output:



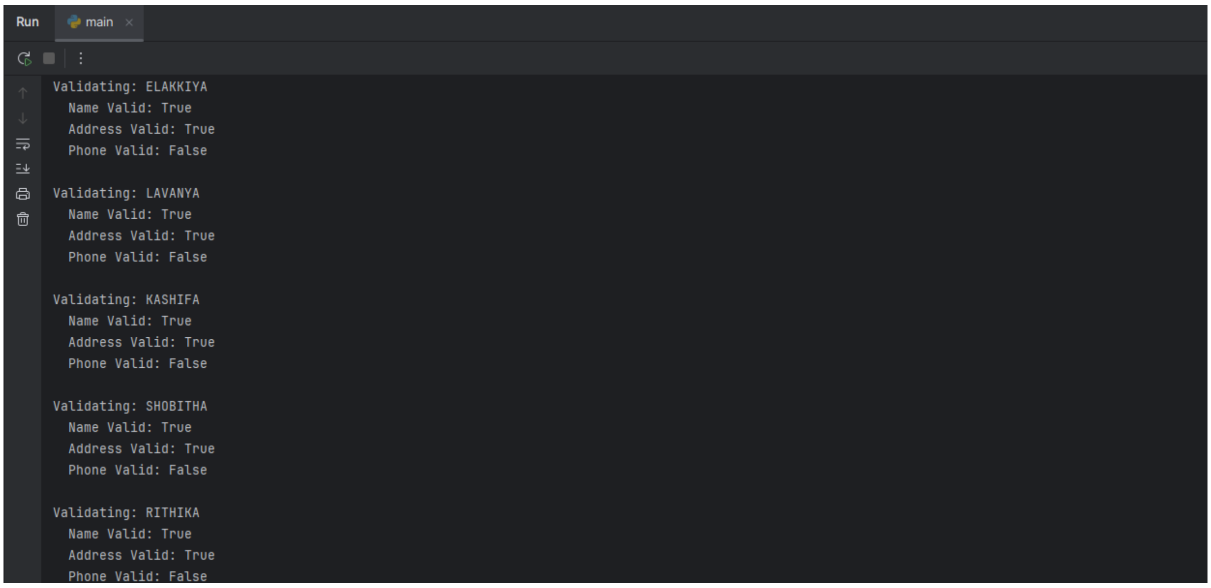
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter your tracking number: TRK002
Tracking Status for TRK002: Out for Delivery

Process finished with exit code 0
```

10. Customer Data Validation: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name addtress or phone number.Validate customer information based on following critirea. Ensure that names contain only letters and are properly capitalized, addresses do not contain special characters, and phone numbers follow a specific format (e.g., ###-###-####).



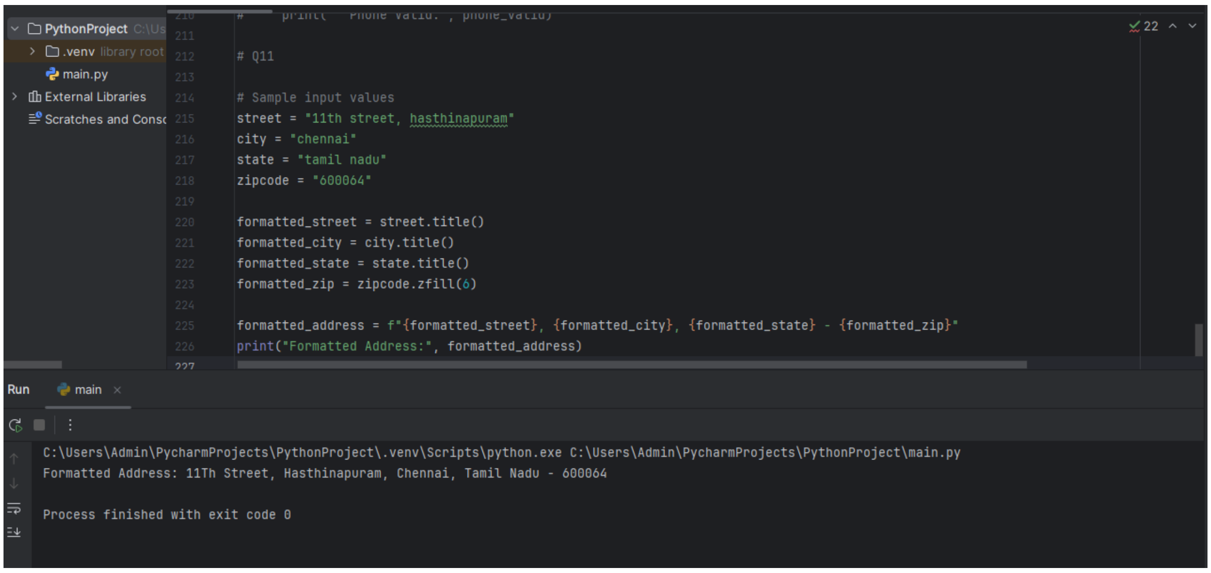
```python
# Q10

cursor.execute("SELECT NAME, ADDRESS, CONTACTNUMBER FROM USER")
users = cursor.fetchall()

for name, address, phone in users:
    print(f"\nValidating: {name}")

    name_valid = name.replace(" ", "").isalpha()

    address_valid = any(c.isalnum() for c in address)

    phone_valid = (
        len(phone) == 12 and
        phone[3] == "-" and
        phone[7] == "-" and
        phone.replace("-", "").isdigit()
    )

    print("  Name Valid:", name_valid)
    print("  Address Valid:", address_valid)
    print("  Phone Valid:", phone_valid)
```

Output:



```
Validating: ELAKKIYA
  Name Valid: True
  Address Valid: True
  Phone Valid: False

Validating: LAVANYA
  Name Valid: True
  Address Valid: True
  Phone Valid: False

Validating: KASHIFA
  Name Valid: True
  Address Valid: True
  Phone Valid: False

Validating: SHOBITHA
  Name Valid: True
  Address Valid: True
  Phone Valid: False

Validating: RITHIKA
  Name Valid: True
  Address Valid: True
  Phone Valid: False
```

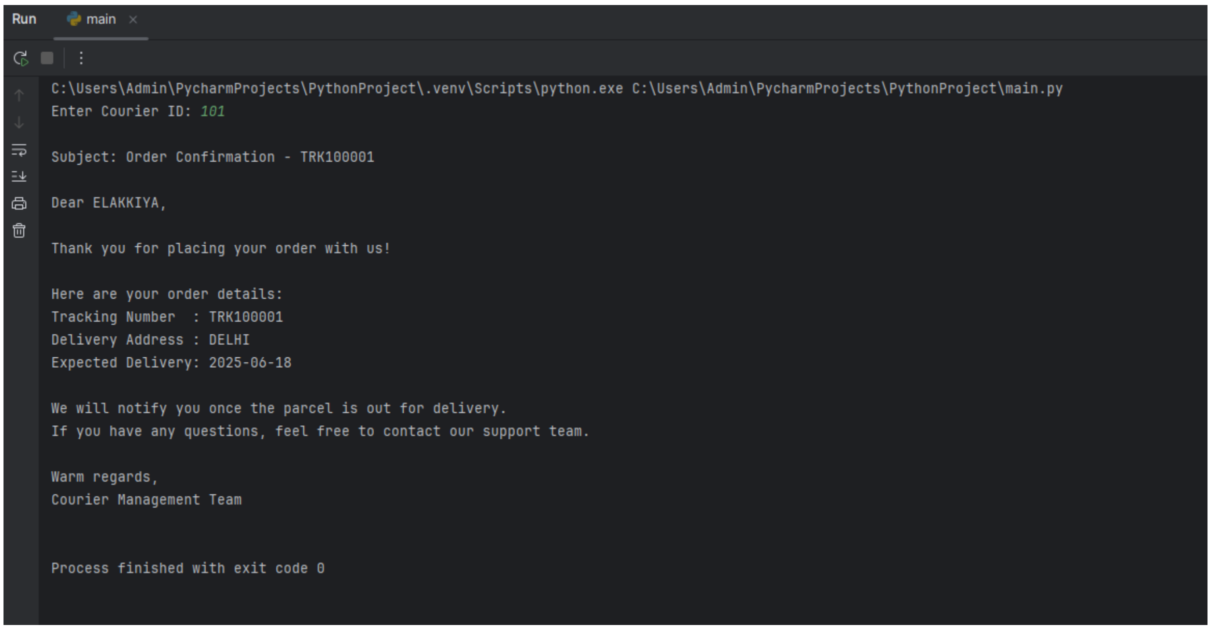
11.Address Formatting: Develop a function that takes an address as input (street, city, state, zip code) and formats it correctly, including capitalizing the first letter of each word and properly formatting the zip code.



```
#    print("Phone Valid:", phone_valid)

# Q11

# Sample input values
street = "11th street, hasthinapuram"
city = "chennai"
state = "tamil nadu"
zipcode = "600064"

formatted_street = street.title()
formatted_city = city.title()
formatted_state = state.title()
formatted_zip = zipcode.zfill(6)

formatted_address = f"{formatted_street}, {formatted_city}, {formatted_state} - {formatted_zip}"
print("Formatted Address:", formatted_address)
```

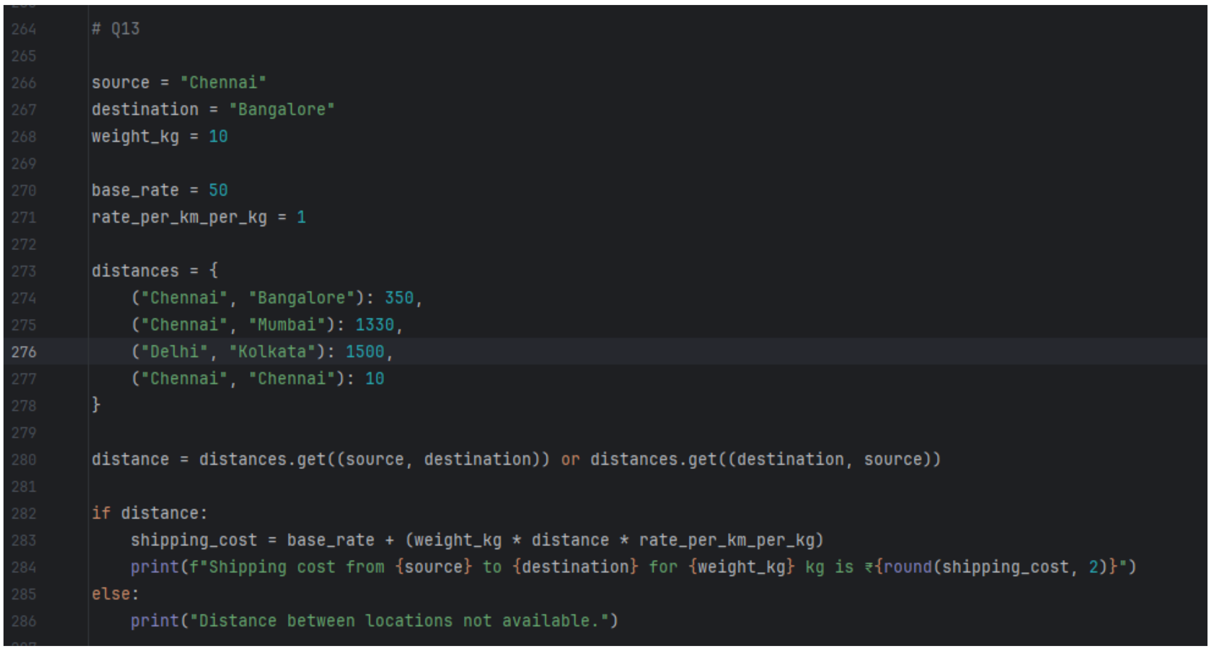
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Formatted Address: 11Th Street, Hasthinapuram, Chennai, Tamil Nadu - 600064

Process finished with exit code 0
```

12.Order Confirmation Email: Create a program that generates an order confirmation email. The email should include details such as the customer's name, order number, delivery address, and expected delivery date.



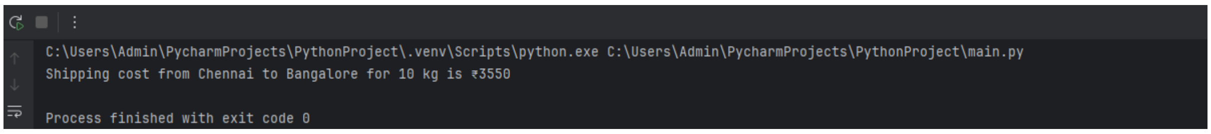
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter Courier ID: 101

Subject: Order Confirmation - TRK100001

Dear ELAKKIYA,

Thank you for placing your order with us!

Here are your order details:
Tracking Number  : TRK100001
Delivery Address : DELHI
Expected Delivery: 2025-06-18

We will notify you once the parcel is out for delivery.
If you have any questions, feel free to contact our support team.

Warm regards,
Courier Management Team


Process finished with exit code 0
```

13. Calculate Shipping Costs: Develop a function that calculates the shipping cost based on the distance between two locations and the weight of the parcel. You can use string inputs for the source and destination addresses.



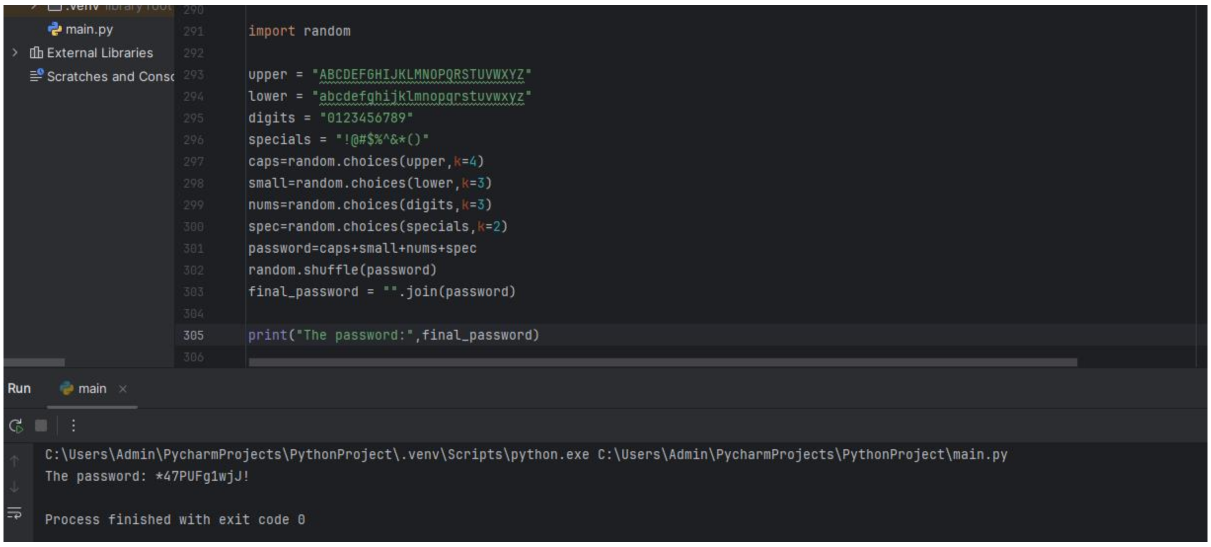
```python
# Q13

source = "Chennai"
destination = "Bangalore"
weight_kg = 10

base_rate = 50
rate_per_km_per_kg = 1

distances = {
    ("Chennai", "Bangalore"): 350,
    ("Chennai", "Mumbai"): 1330,
    ("Delhi", "Kolkata"): 1500,
    ("Chennai", "Chennai"): 10
}

distance = distances.get((source, destination)) or distances.get((destination, source))

if distance:
    shipping_cost = base_rate + (weight_kg * distance * rate_per_km_per_kg)
    print(f"Shipping cost from {source} to {destination} for {weight_kg} kg is ₹{round(shipping_cost, 2)}")
else:
    print("Distance between locations not available.")
```

Output:



```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Shipping cost from Chennai to Bangalore for 10 kg is ₹3550

Process finished with exit code 0
```

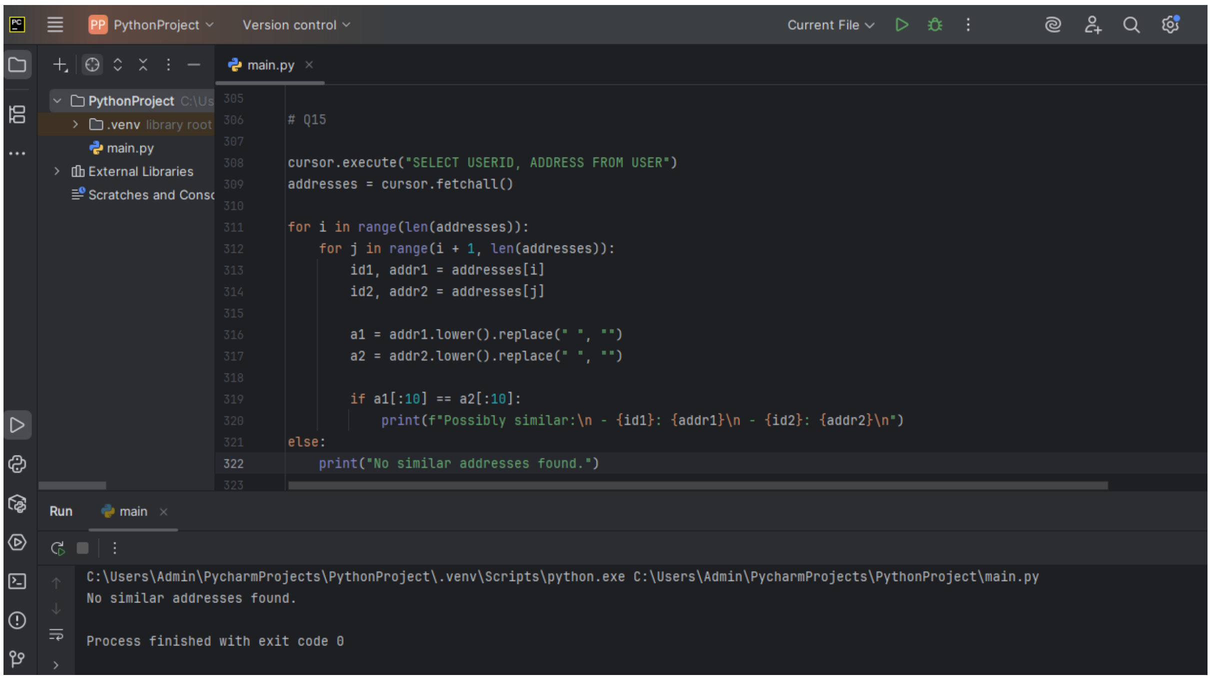
14. Password Generator: Create a function that generates secure passwords for courier system accounts. Ensure the passwords contain a mix of uppercase letters, lowercase letters, numbers, and special characters.



```python
import random

upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
lower = "abcdefghijklmnopqrstuvwxyz"
digits = "0123456789"
specials = "!@#$%^&*()"
caps=random.choices(upper,k=4)
small=random.choices(lower,k=3)
nums=random.choices(digits,k=3)
spec=random.choices(specials,k=2)
password=caps+small+nums+spec
random.shuffle(password)
final_password = "".join(password)

print("The password:",final_password)
```

```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
The password: *47PUFg1wjJ!

Process finished with exit code 0
```

15. Find Similar Addresses: Implement a function that finds similar addresses in the system. This can be useful for identifying duplicate customer entries or optimizing delivery routes.Use string functions to implement this.



```python

# Q15

cursor.execute("SELECT USERID, ADDRESS FROM USER")
addresses = cursor.fetchall()

for i in range(len(addresses)):
    for j in range(i + 1, len(addresses)):
        id1, addr1 = addresses[i]
        id2, addr2 = addresses[j]

        a1 = addr1.lower().replace(" ", "")
        a2 = addr2.lower().replace(" ", "")

        if a1[:10] == a2[:10]:
            print(f"Possibly similar:\n - {id1}: {addr1}\n - {id2}: {addr2}\n")
else:
    print("No similar addresses found.")
```

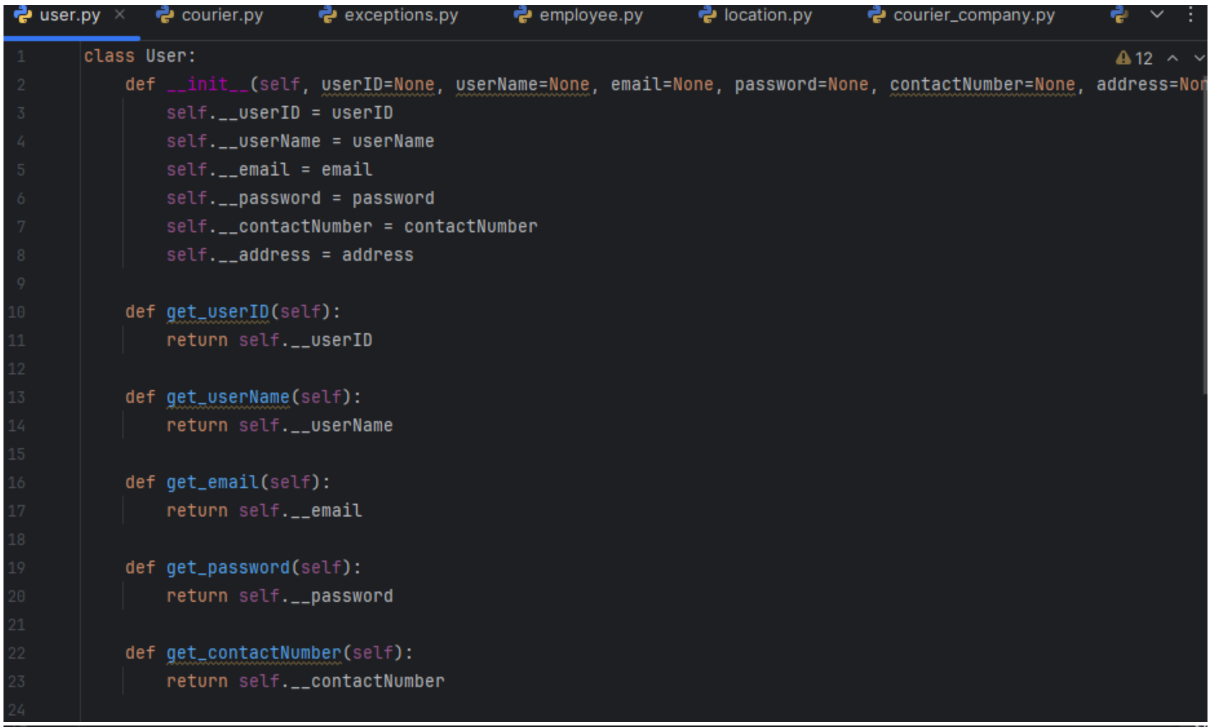
```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
No similar addresses found.

Process finished with exit code 0
```

Task 5 on next page…

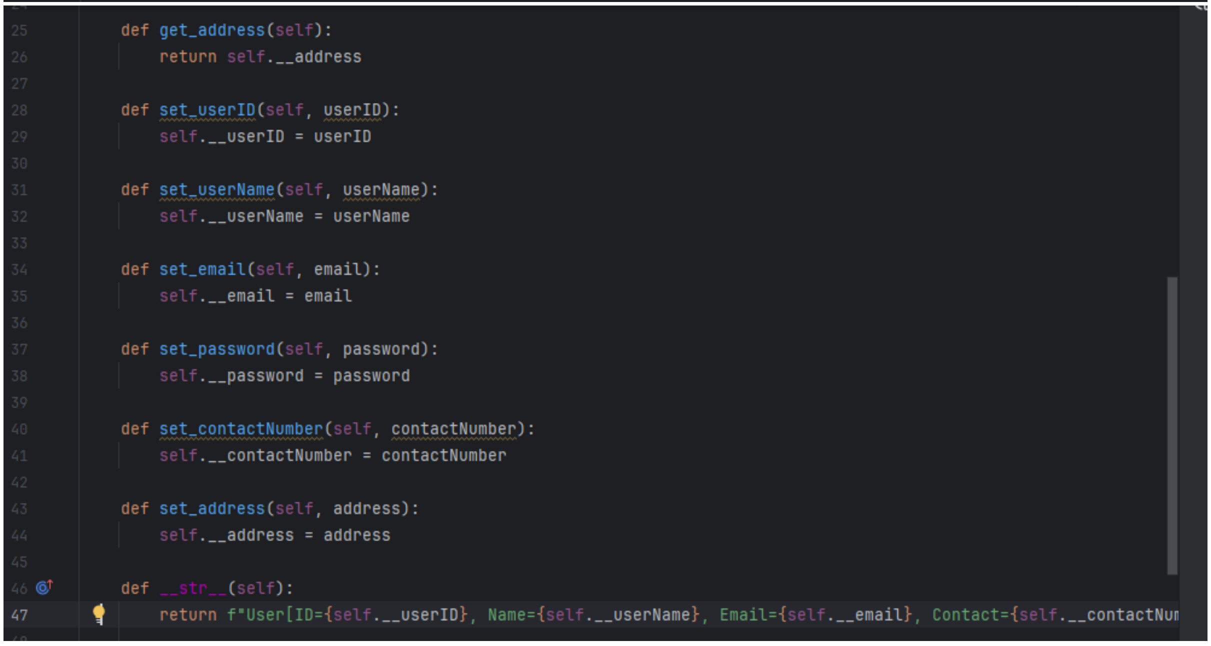
Task 5:

Object Oriented Programming Scope : Entity classes/Models/POJO, Abstraction/Encapsulation Create the following model/entity classes within package entities with variables declared private, constructors(default and parametrized,getters,setters and toString())

1. User Class:
   Variables:
   userID , userName , email , password , contactNumber , address

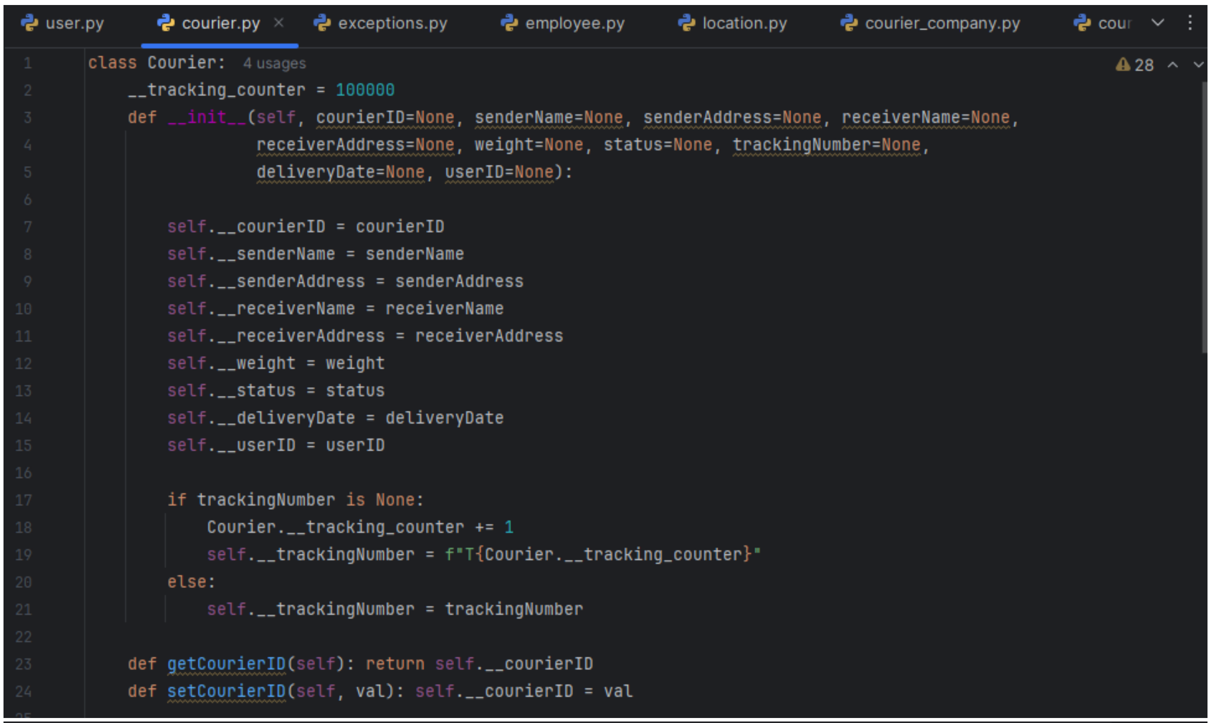


```python
class User:
    def __init__(self, userID=None, userName=None, email=None, password=None, contactNumber=None, address=Nor
        self.__userID = userID
        self.__userName = userName
        self.__email = email
        self.__password = password
        self.__contactNumber = contactNumber
        self.__address = address

    def get_userID(self):
        return self.__userID

    def get_userName(self):
        return self.__userName

    def get_email(self):
        return self.__email

    def get_password(self):
        return self.__password

    def get_contactNumber(self):
        return self.__contactNumber

```

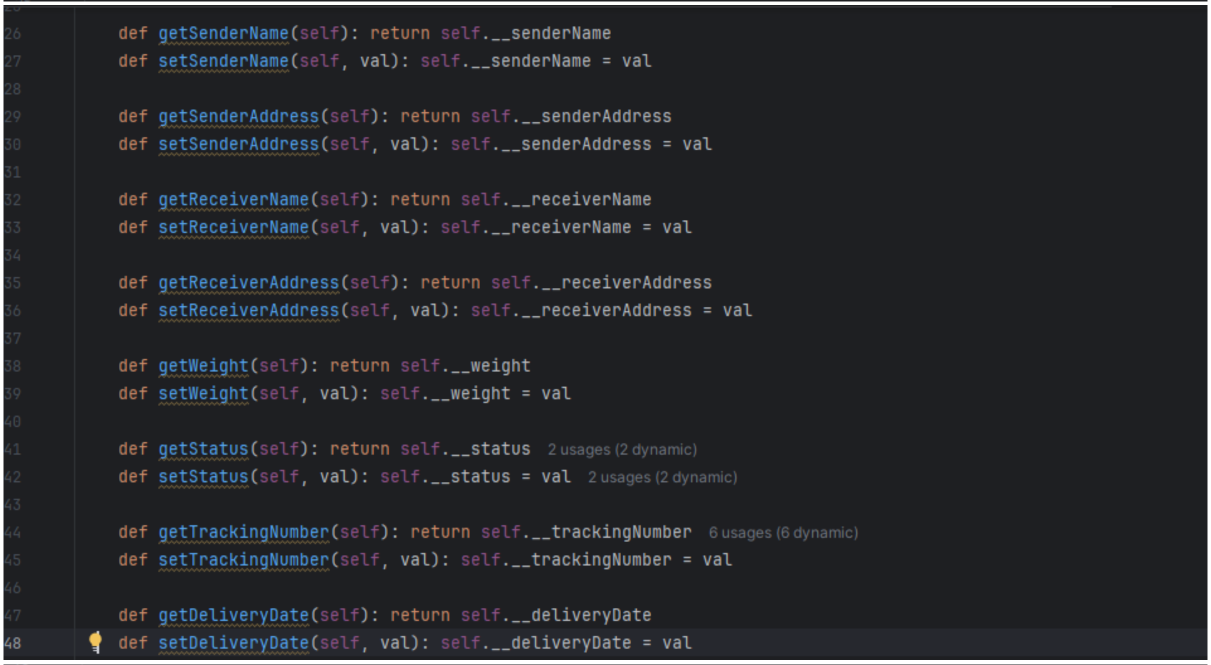


```python
    def get_address(self):
        return self.__address

    def set_userID(self, userID):
        self.__userID = userID

    def set_userName(self, userName):
        self.__userName = userName

    def set_email(self, email):
        self.__email = email

    def set_password(self, password):
        self.__password = password

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def set_address(self, address):
        self.__address = address

    def __str__(self):
        return f"User[ID={self.__userID}, Name={self.__userName}, Email={self.__email}, Contact={self.__contactNum
```

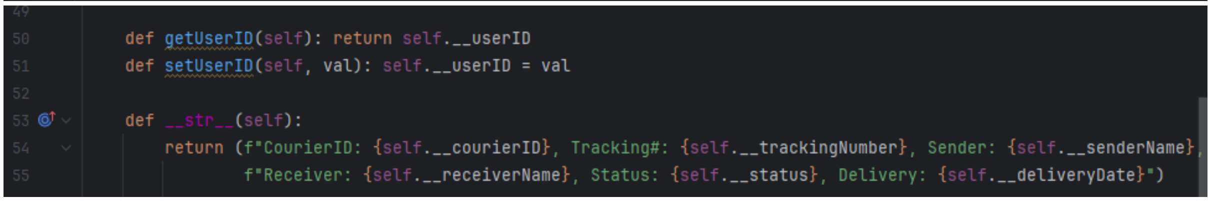
2. Courier Class
   Variables:
   courierID , senderName , senderAddress , receiverName , receiverAddress , weight , status, trackingNumber , deliveryDate ,userId



```python
class Courier:
    __tracking_counter = 100000
    def __init__(self, courierID=None, senderName=None, senderAddress=None, receiverName=None,
                 receiverAddress=None, weight=None, status=None, trackingNumber=None,
                 deliveryDate=None, userID=None):

        self.__courierID = courierID
        self.__senderName = senderName
        self.__senderAddress = senderAddress
        self.__receiverName = receiverName
        self.__receiverAddress = receiverAddress
        self.__weight = weight
        self.__status = status
        self.__deliveryDate = deliveryDate
        self.__userID = userID

        if trackingNumber is None:
            Courier.__tracking_counter += 1
            self.__trackingNumber = f"T{Courier.__tracking_counter}"
        else:
            self.__trackingNumber = trackingNumber

    def getCourierID(self): return self.__courierID
    def setCourierID(self, val): self.__courierID = val
```

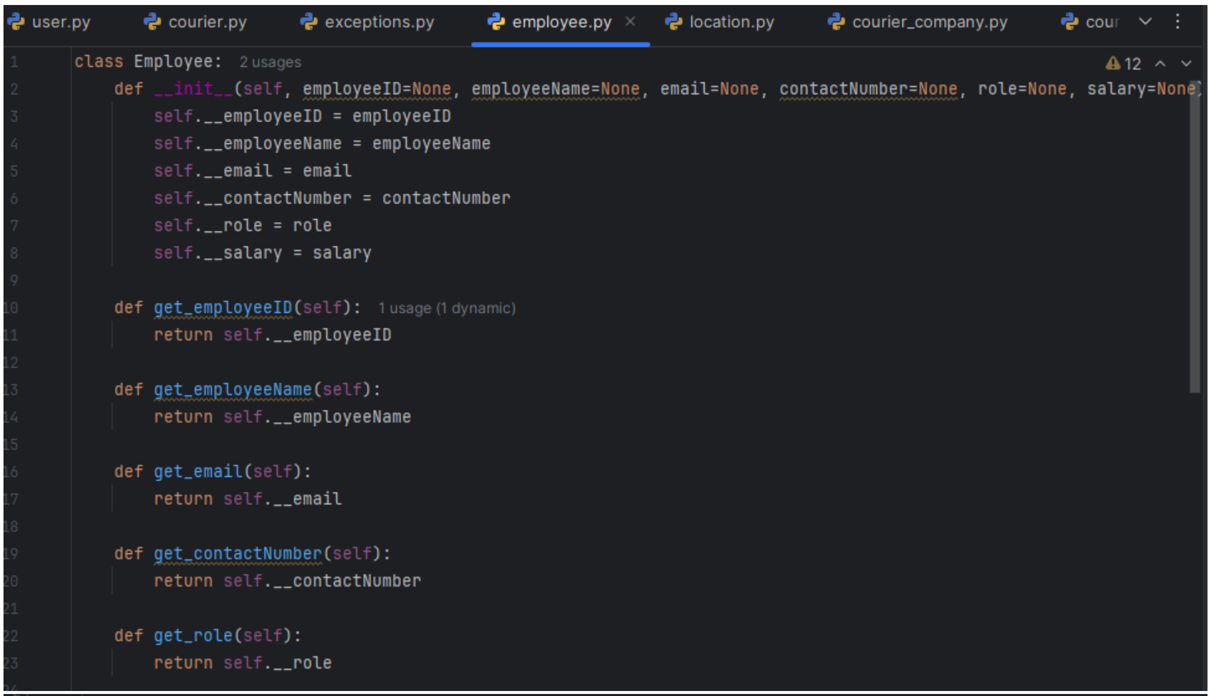


```python
    def getSenderName(self): return self.__senderName
    def setSenderName(self, val): self.__senderName = val

    def getSenderAddress(self): return self.__senderAddress
    def setSenderAddress(self, val): self.__senderAddress = val

    def getReceiverName(self): return self.__receiverName
    def setReceiverName(self, val): self.__receiverName = val

    def getReceiverAddress(self): return self.__receiverAddress
    def setReceiverAddress(self, val): self.__receiverAddress = val

    def getWeight(self): return self.__weight
    def setWeight(self, val): self.__weight = val

    def getStatus(self): return self.__status
    def setStatus(self, val): self.__status = val

    def getTrackingNumber(self): return self.__trackingNumber
    def setTrackingNumber(self, val): self.__trackingNumber = val

    def getDeliveryDate(self): return self.__deliveryDate
    def setDeliveryDate(self, val): self.__deliveryDate = val
```

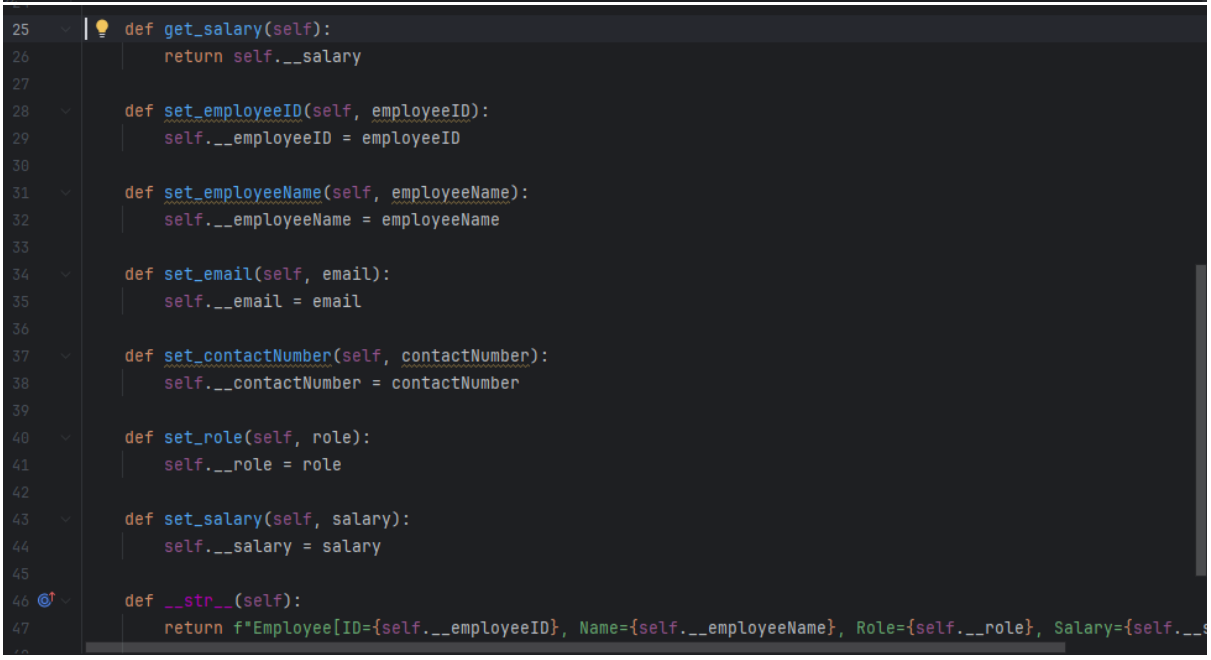


```python
    def getUserID(self): return self.__userID
    def setUserID(self, val): self.__userID = val

    def __str__(self):
        return (f"CourierID: {self.__courierID}, Tracking#: {self.__trackingNumber}, Sender: {self.__senderName}
                f"Receiver: {self.__receiverName}, Status: {self.__status}, Delivery: {self.__deliveryDate}")
```

3. Employee Class:
   Variables:
   employeeID , employeeName , email , contactNumber , role String, salary

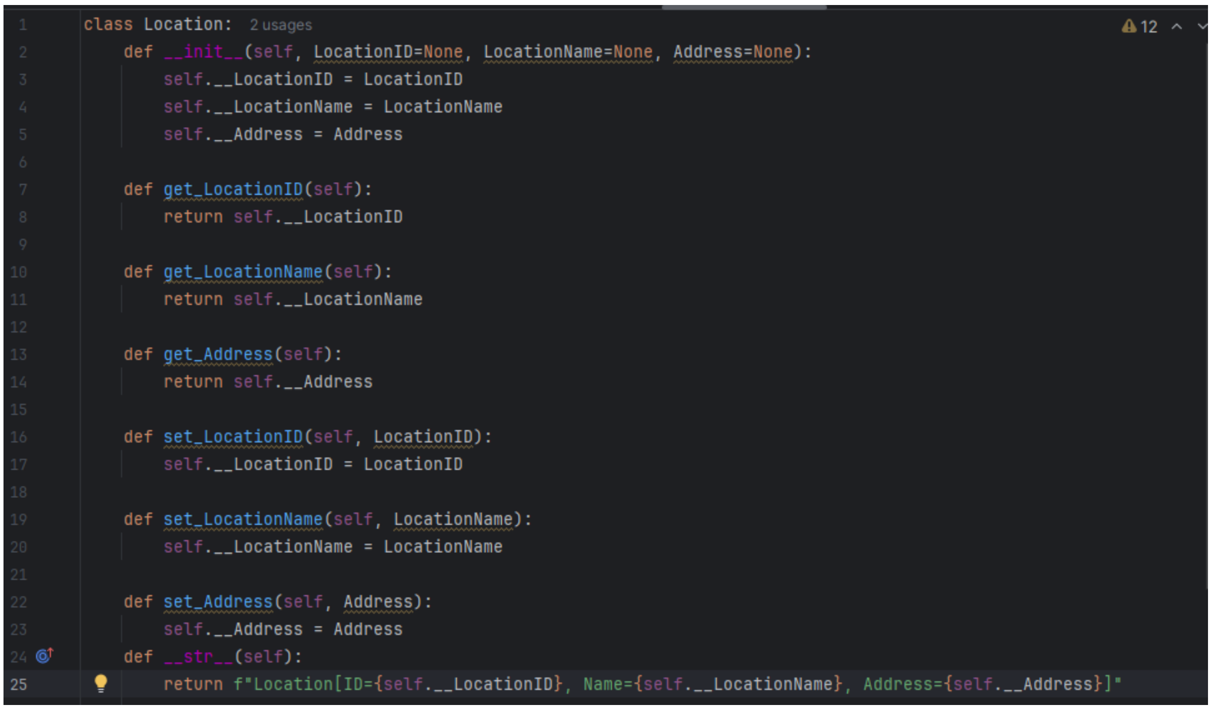


```python
class Employee:
    def __init__(self, employeeID=None, employeeName=None, email=None, contactNumber=None, role=None, salary=None
        self.__employeeID = employeeID
        self.__employeeName = employeeName
        self.__email = email
        self.__contactNumber = contactNumber
        self.__role = role
        self.__salary = salary

    def get_employeeID(self):
        return self.__employeeID

    def get_employeeName(self):
        return self.__employeeName

    def get_email(self):
        return self.__email

    def get_contactNumber(self):
        return self.__contactNumber

    def get_role(self):
        return self.__role
```



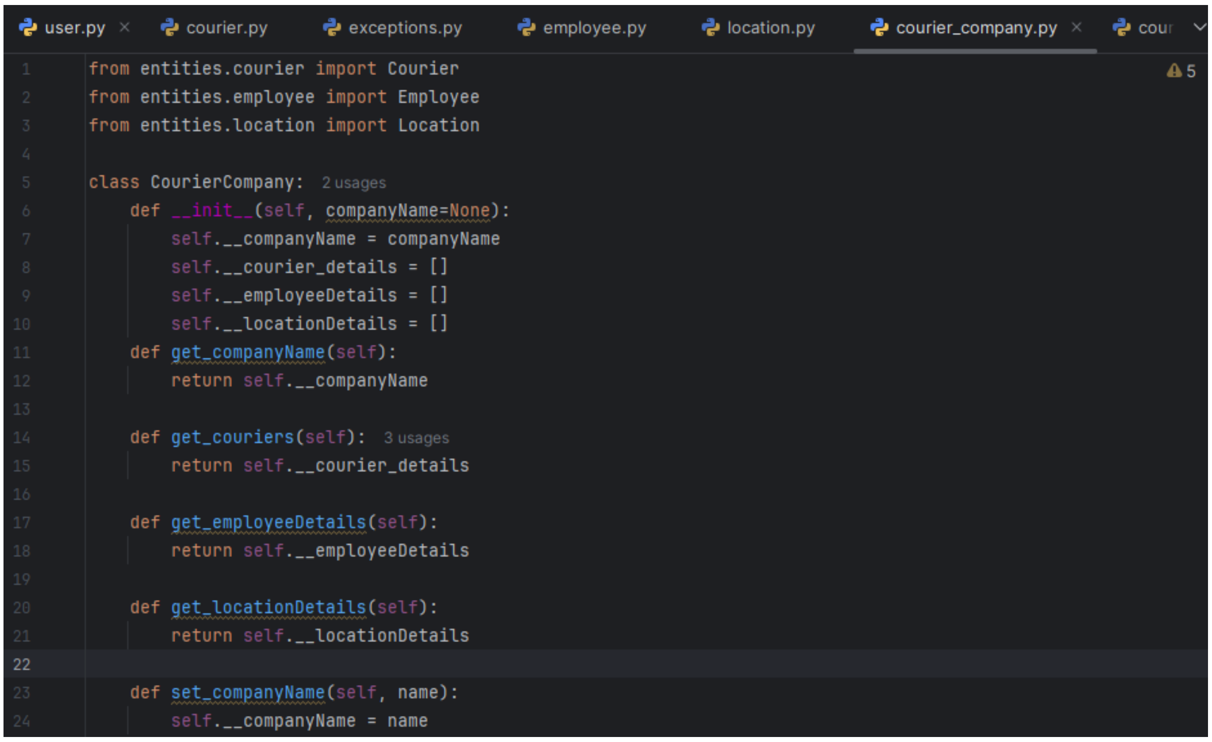
```python
    def get_salary(self):
        return self.__salary

    def set_employeeID(self, employeeID):
        self.__employeeID = employeeID

    def set_employeeName(self, employeeName):
        self.__employeeName = employeeName

    def set_email(self, email):
        self.__email = email

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def set_role(self, role):
        self.__role = role

    def set_salary(self, salary):
        self.__salary = salary

    def __str__(self):
        return f"Employee[ID={self.__employeeID}, Name={self.__employeeName}, Role={self.__role}, Salary={self.__s
```

4. Location Class
   Variables:
   LocationID , LocationName , Address

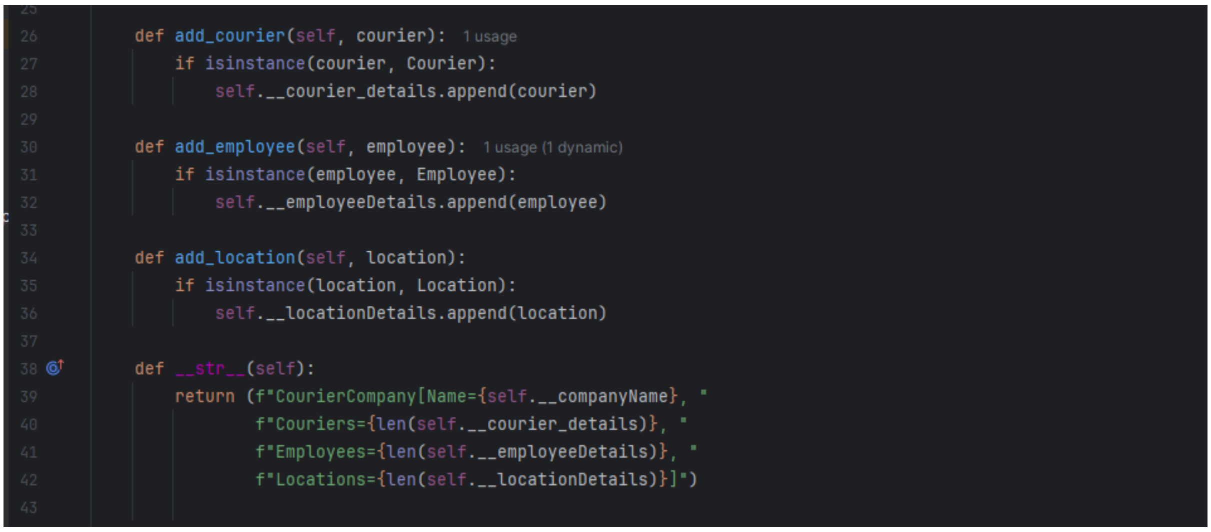


```python
class Location:  2 usages
    def __init__(self, LocationID=None, LocationName=None, Address=None):
        self.__LocationID = LocationID
        self.__LocationName = LocationName
        self.__Address = Address

    def get_LocationID(self):
        return self.__LocationID

    def get_LocationName(self):
        return self.__LocationName

    def get_Address(self):
        return self.__Address

    def set_LocationID(self, LocationID):
        self.__LocationID = LocationID

    def set_LocationName(self, LocationName):
        self.__LocationName = LocationName

    def set_Address(self, Address):
        self.__Address = Address
    def __str__(self):
        return f"Location[ID={self.__LocationID}, Name={self.__LocationName}, Address={self.__Address}]"
```

5. CourierCompany Class
   Variables:
   companyName , courierDetails -collection of Courier Objects, employeeDetails- collection of Employee Objects, locationDetails - collection of Location Objects.

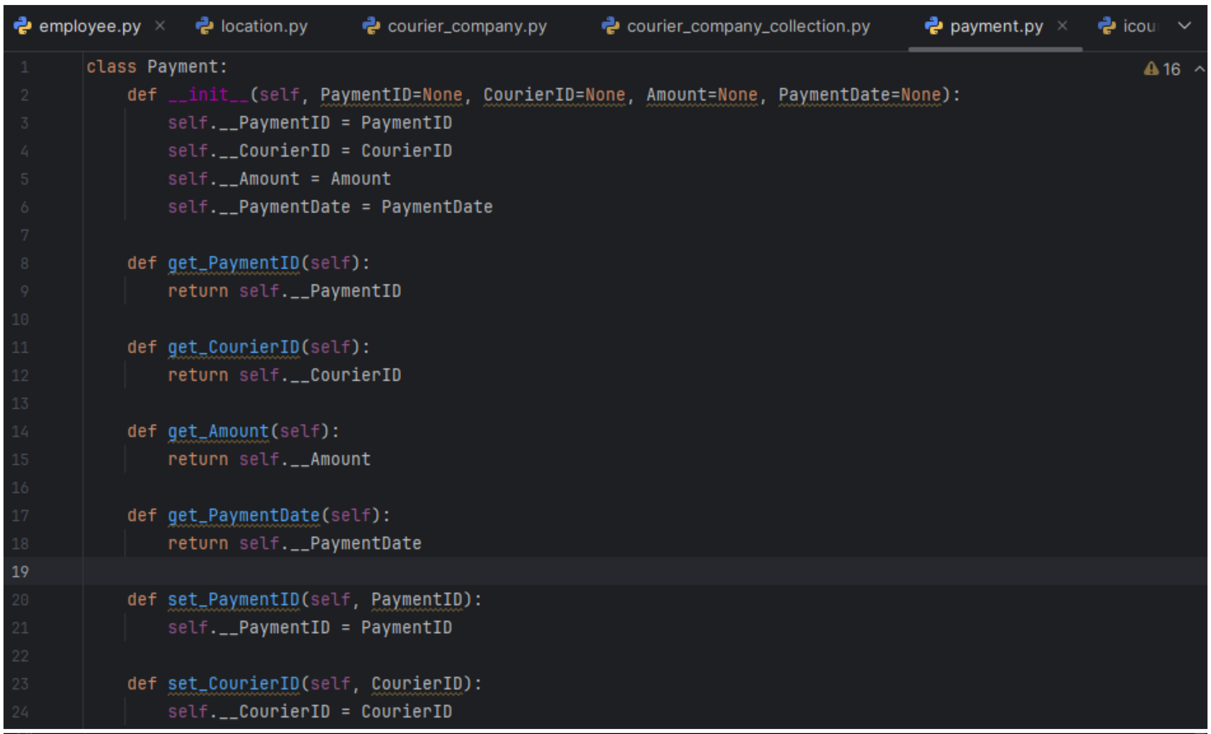


```python
from entities.courier import Courier
from entities.employee import Employee
from entities.location import Location

class CourierCompany:  2 usages
    def __init__(self, companyName=None):
        self.__companyName = companyName
        self.__courier_details = []
        self.__employeeDetails = []
        self.__locationDetails = []
    def get_companyName(self):
        return self.__companyName

    def get_couriers(self):  3 usages
        return self.__courier_details

    def get_employeeDetails(self):
        return self.__employeeDetails

    def get_locationDetails(self):
        return self.__locationDetails

    def set_companyName(self, name):
        self.__companyName = name
```

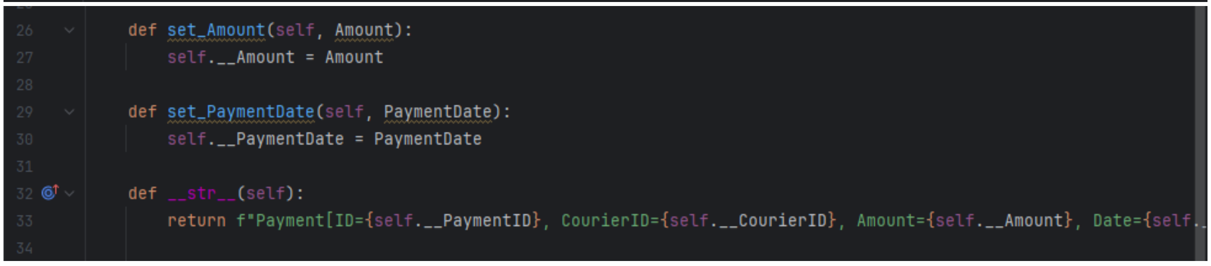


```python
    def add_courier(self, courier):
        if isinstance(courier, Courier):
            self.__courier_details.append(courier)

    def add_employee(self, employee):
        if isinstance(employee, Employee):
            self.__employeeDetails.append(employee)

    def add_location(self, location):
        if isinstance(location, Location):
            self.__locationDetails.append(location)

    def __str__(self):
        return (f"CourierCompany[Name={self.__companyName}, "
                f"Couriers={len(self.__courier_details)}, "
                f"Employees={len(self.__employeeDetails)}, "
                f"Locations={len(self.__locationDetails)}]")
```

6. Payment Class:
   Variables:
   PaymentID long, CourierID long, Amount double, PaymentDate Date



```python
class Payment:
    def __init__(self, PaymentID=None, CourierID=None, Amount=None, PaymentDate=None):
        self.__PaymentID = PaymentID
        self.__CourierID = CourierID
        self.__Amount = Amount
        self.__PaymentDate = PaymentDate

    def get_PaymentID(self):
        return self.__PaymentID

    def get_CourierID(self):
        return self.__CourierID

    def get_Amount(self):
        return self.__Amount

    def get_PaymentDate(self):
        return self.__PaymentDate

    def set_PaymentID(self, PaymentID):
        self.__PaymentID = PaymentID

    def set_CourierID(self, CourierID):
        self.__CourierID = CourierID
```



```python
    def set_Amount(self, Amount):
        self.__Amount = Amount

    def set_PaymentDate(self, PaymentDate):
        self.__PaymentDate = PaymentDate

    def __str__(self):
        return f"Payment[ID={self.__PaymentID}, CourierID={self.__CourierID}, Amount={self.__Amount}, Date={self.
```

Task 6: Service Provider Interface /Abstract class

Create 2 Interface /Abstract class ICourierUserService and ICourierAdminService interface ICourierUserService {

// Customer-related functions

placeOrder()

/** Place a new courier order.

* @param courierObj Courier object created using values entered by users

* @return The unique tracking number for the courier order .

Use a static variable to generate unique tracking number. Initialize the static variable in Courier class with some random value. Increment the static variable each time in the constructor to generate next values.

getOrderStatus();

/**Get the status of a courier order.

*@param trackingNumber The tracking number of the courier order.

* @return The status of the courier order (e.g., yetToTransit, In Transit, Delivered).

*/

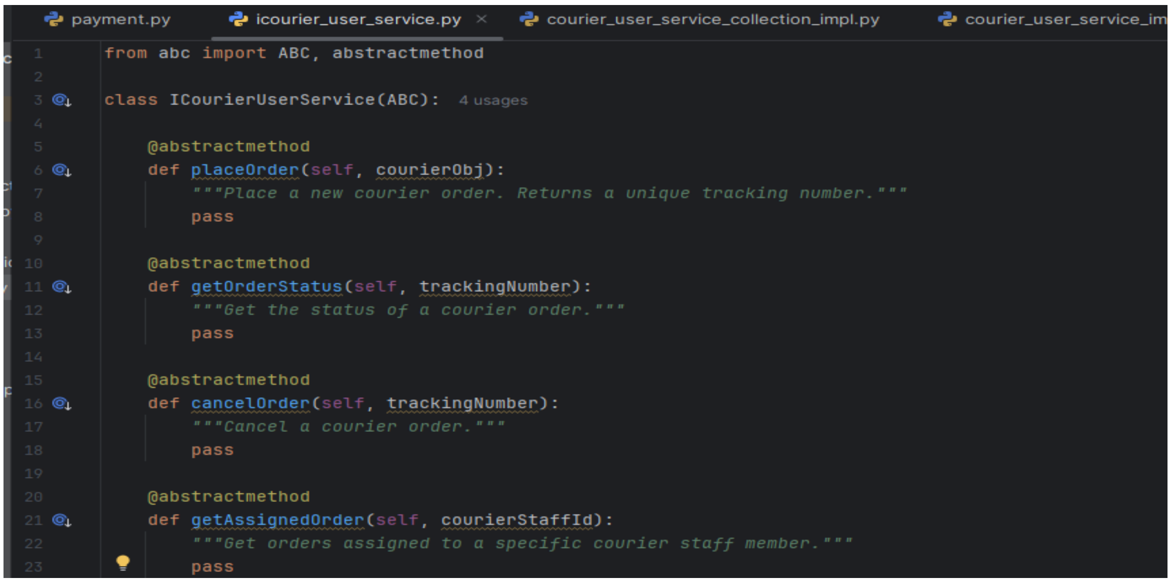
cancelOrder()

/** Cancel a courier order.

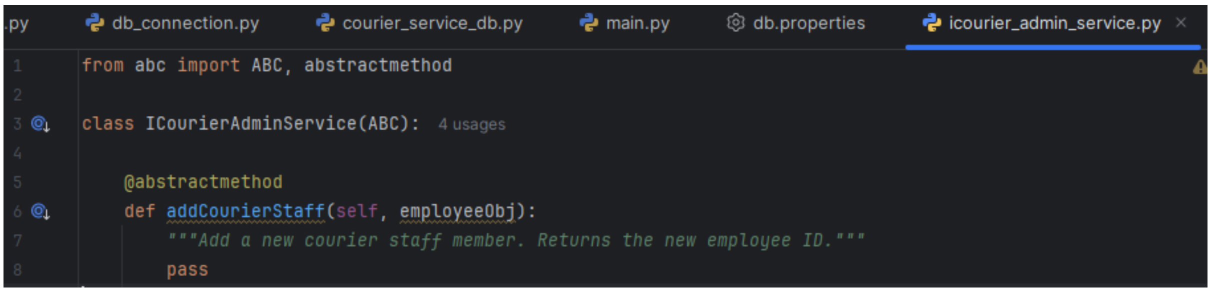
* @param trackingNumber The tracking number of the courier order to be canceled.

* @return True if the order was successfully canceled, false otherwise.

*/



```python
from abc import ABC, abstractmethod

class ICourierUserService(ABC):

    @abstractmethod
    def placeOrder(self, courierObj):
        """Place a new courier order. Returns a unique tracking number."""
        pass

    @abstractmethod
    def getOrderStatus(self, trackingNumber):
        """Get the status of a courier order."""
        pass

    @abstractmethod
    def cancelOrder(self, trackingNumber):
        """Cancel a courier order."""
        pass

    @abstractmethod
    def getAssignedOrder(self, courierStaffId):
        """Get orders assigned to a specific courier staff member."""
        pass
```



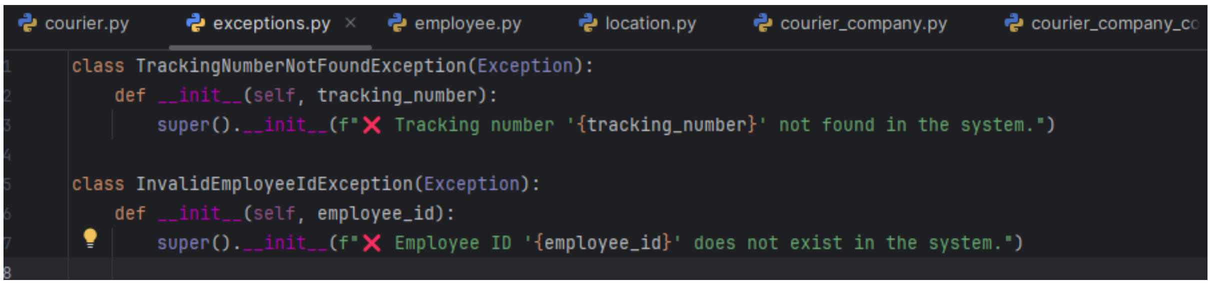
```
from abc import ABC, abstractmethod

class ICourierAdminService(ABC):  4 usages

    @abstractmethod
    def addCourierStaff(self, employeeObj):
        """Add a new courier staff member. Returns the new employee ID."""
        pass
```

Task 7: Exception Handling

(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage) Define the following custom exceptions and throw them in methods whenever needed . Handle all the excpetionsin main method,

1. TrackingNumberNotFoundException :throw this exception when user try to withdraw amount or transfer amount to another account
2. InvalidEmployeeIdException throw this exception when id entered for the employee not existing in the system

   Exception 1 and 2 are coded together in exceptions.py file

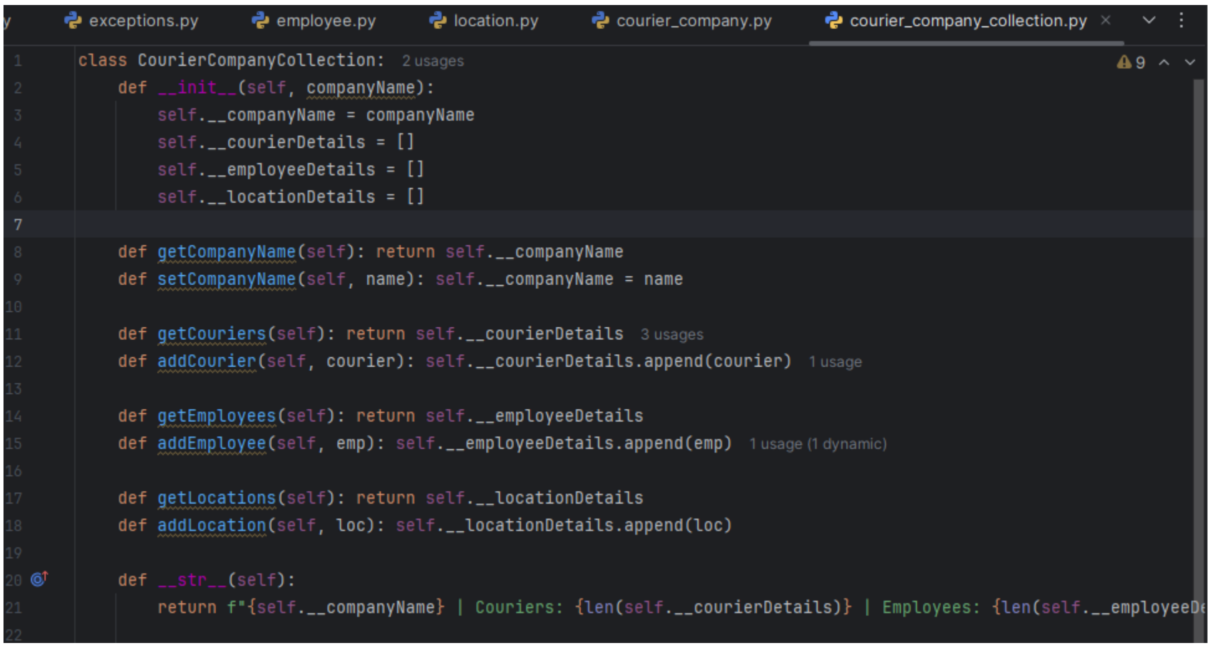


```
class TrackingNumberNotFoundException(Exception):
    def __init__(self, tracking_number):
        super().__init__(f"❌ Tracking number '{tracking_number}' not found in the system.")

class InvalidEmployeeIdException(Exception):
    def __init__(self, employee_id):
        super().__init__(f"❌ Employee ID '{employee_id}' does not exist in the system.")
```

Task 8: Collections Scope:

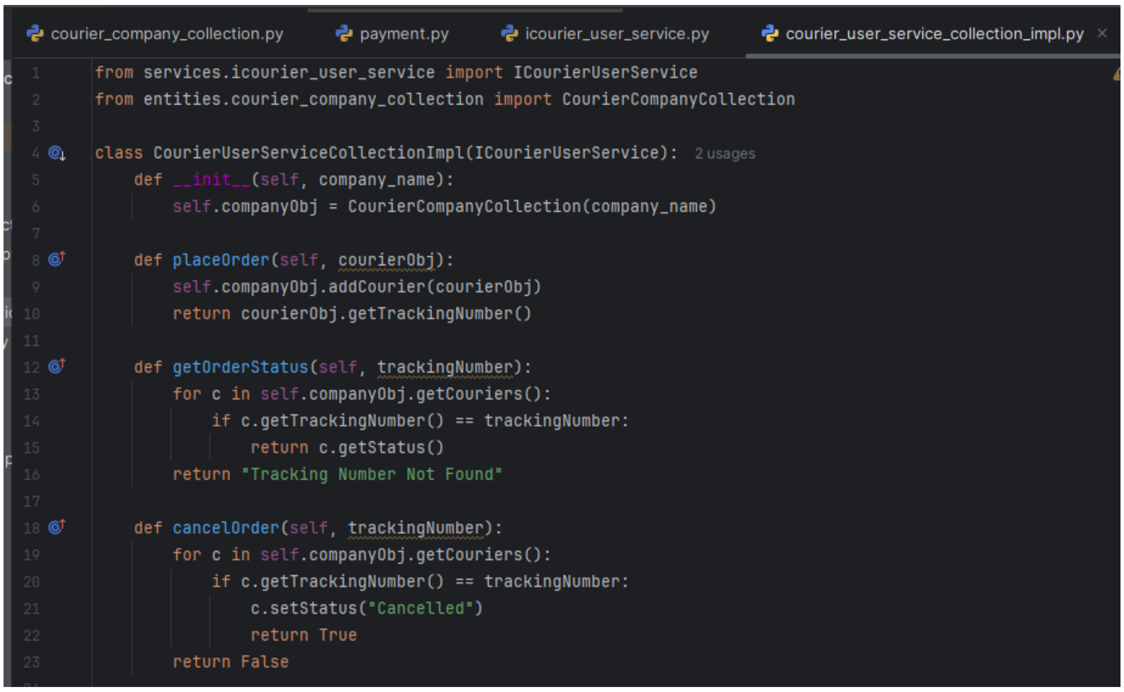
ArrayList/Hashmap Task: Improve the Courier Management System by using Java collections:

1. Create a new model named CourierCompanyCollection in entity package replacing the Array of Objects with List to accommodate dynamic updates in the CourierCompany class

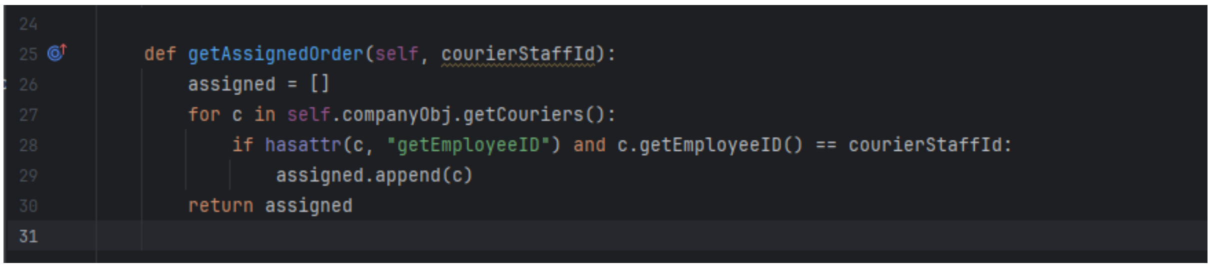


```python
class CourierCompanyCollection:
    def __init__(self, companyName):
        self.__companyName = companyName
        self.__courierDetails = []
        self.__employeeDetails = []
        self.__locationDetails = []

    def getCompanyName(self): return self.__companyName
    def setCompanyName(self, name): self.__companyName = name

    def getCouriers(self): return self.__courierDetails
    def addCourier(self, courier): self.__courierDetails.append(courier)

    def getEmployees(self): return self.__employeeDetails
    def addEmployee(self, emp): self.__employeeDetails.append(emp)

    def getLocations(self): return self.__locationDetails
    def addLocation(self, loc): self.__locationDetails.append(loc)

    def __str__(self):
        return f"{self.__companyName} | Couriers: {len(self.__courierDetails)} | Employees: {len(self.__employeeDe
```

2. Create a new implementation class CourierUserServiceCollectionImpl class in package dao which implements ICourierUserService interface which holds a variable named companyObj of type CourierCompanyCollection


courier_company_collection.py
payment.py
icourier_user_service.py
courier_user_service_collection_impl.py

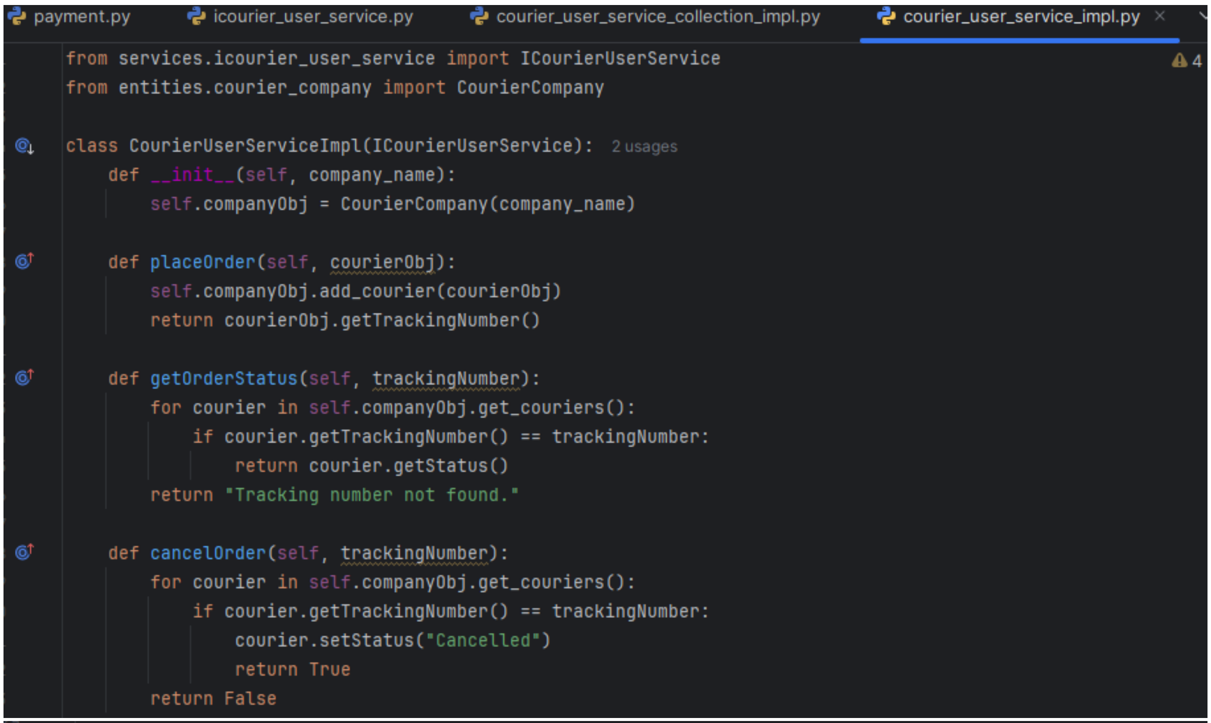
```python
from services.icourier_user_service import ICourierUserService
from entities.courier_company_collection import CourierCompanyCollection

class CourierUserServiceCollectionImpl(ICourierUserService):
    def __init__(self, company_name):
        self.companyObj = CourierCompanyCollection(company_name)

    def placeOrder(self, courierObj):
        self.companyObj.addCourier(courierObj)
        return courierObj.getTrackingNumber()

    def getOrderStatus(self, trackingNumber):
        for c in self.companyObj.getCouriers():
            if c.getTrackingNumber() == trackingNumber:
                return c.getStatus()
        return "Tracking Number Not Found"

    def cancelOrder(self, trackingNumber):
        for c in self.companyObj.getCouriers():
            if c.getTrackingNumber() == trackingNumber:
                c.setStatus("Cancelled")
                return True
        return False
```



```python
    def getAssignedOrder(self, courierStaffId):
        assigned = []
        for c in self.companyObj.getCouriers():
            if hasattr(c, "getEmployeeID") and c.getEmployeeID() == courierStaffId:
                assigned.append(c)
        return assigned
```

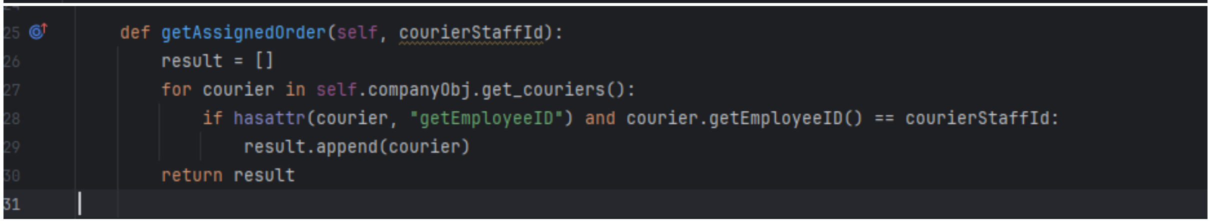
Task 9: Service implementation

1. Create CourierUserServiceImpl class which implements ICourierUserService interface which holds a variable named companyObj of type CourierCompany. This variable can be used to access the Object Arrays to access data relevant in method implementations.



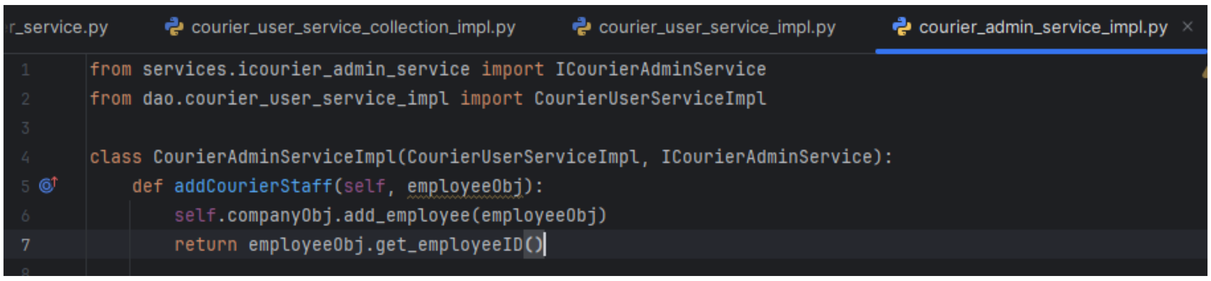
payment.py icourier_user_service.py courier_user_service_collection_impl.py courier_user_service_impl.py

```python
from services.icourier_user_service import ICourierUserService
from entities.courier_company import CourierCompany

class CourierUserServiceImpl(ICourierUserService):  2 usages
    def __init__(self, company_name):
        self.companyObj = CourierCompany(company_name)

    def placeOrder(self, courierObj):
        self.companyObj.add_courier(courierObj)
        return courierObj.getTrackingNumber()

    def getOrderStatus(self, trackingNumber):
        for courier in self.companyObj.get_couriers():
            if courier.getTrackingNumber() == trackingNumber:
                return courier.getStatus()
        return "Tracking number not found."

    def cancelOrder(self, trackingNumber):
        for courier in self.companyObj.get_couriers():
            if courier.getTrackingNumber() == trackingNumber:
                courier.setStatus("Cancelled")
                return True
        return False
```



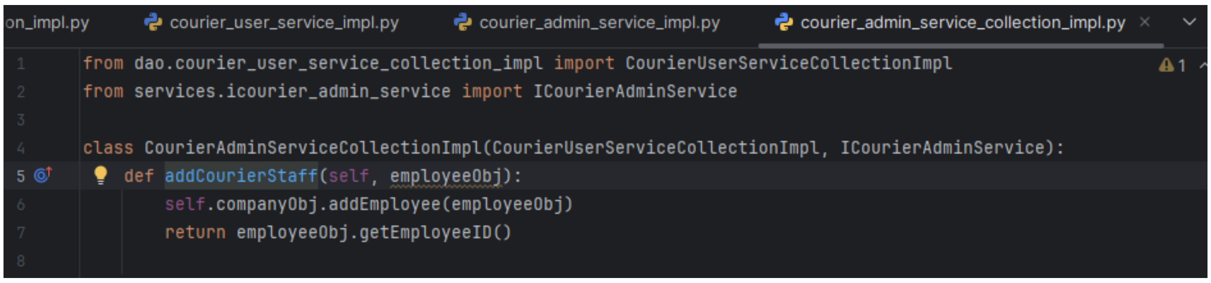
```python
    def getAssignedOrder(self, courierStaffId):
        result = []
        for courier in self.companyObj.get_couriers():
            if hasattr(courier, "getEmployeeID") and courier.getEmployeeID() == courierStaffId:
                result.append(courier)
        return result
```

2. Create CourierAdminService Impl class which inherits from CourierUserServiceImpl and implements ICourierAdminService interface.


r_service.py | courier_user_service_collection_impl.py | courier_user_service_impl.py | courier_admin_service_impl.py

```python
from services.icourier_admin_service import ICourierAdminService
from dao.courier_user_service_impl import CourierUserServiceImpl

class CourierAdminServiceImpl(CourierUserServiceImpl, ICourierAdminService):
    def addCourierStaff(self, employeeObj):
        self.companyObj.add_employee(employeeObj)
        return employeeObj.get_employeeID()
```

3. Create CourierAdminServiceCollectionImpl class which inherits from CourierUserServiceColectionImpl and implements ICourierAdminService interface.

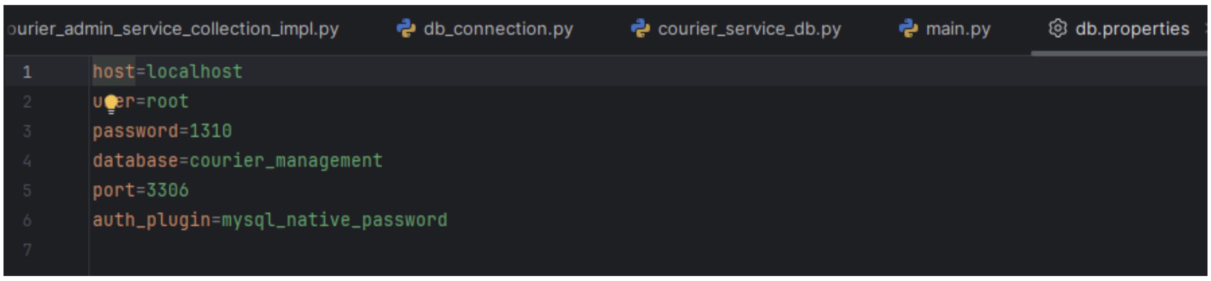

on_impl.py | courier_user_service_impl.py | courier_admin_service_impl.py | courier_admin_service_collection_impl.py

```python
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
from services.icourier_admin_service import ICourierAdminService

class CourierAdminServiceCollectionImpl(CourierUserServiceCollectionImpl, ICourierAdminService):
    def addCourierStaff(self, employeeObj):
        self.companyObj.addEmployee(employeeObj)
        return employeeObj.getEmployeeID()
```

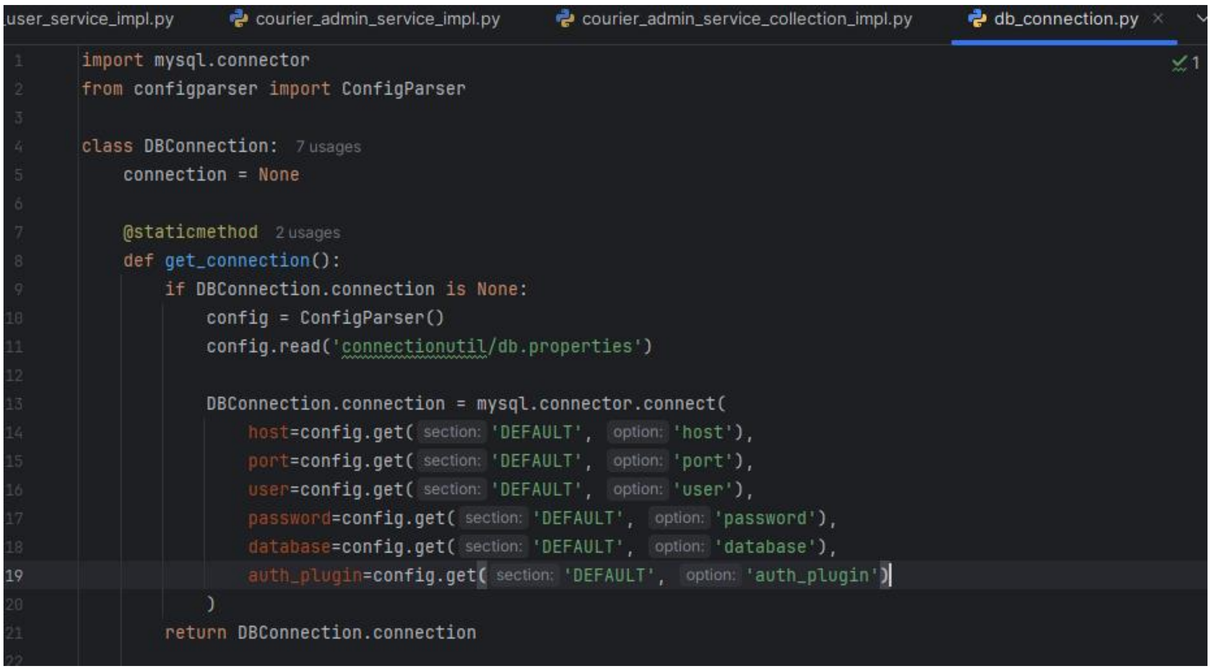
Task 10: Database Interaction

Connect your application to the SQL database for the Courier Management System

1. Write code to establish a connection to your SQL database. Create a class DBConnection in a package connectionutil with a static variable connection of Type Connection and a static method getConnection() which returns connection.
   Connection properties supplied in the connection string should be read from a property file.

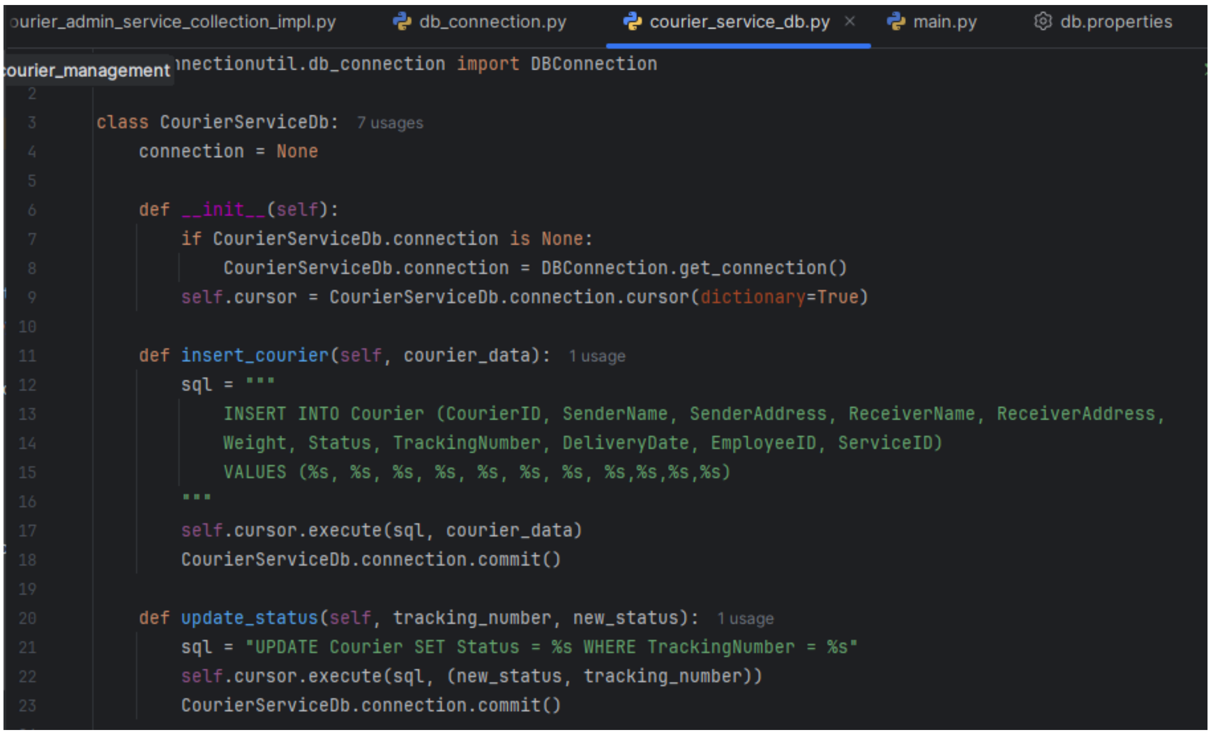

ourier_admin_service_collection_impl.py | db_connection.py | courier_service_db.py | main.py | db.properties

```
host=localhost
user=root
password=1310
database=courier_management
port=3306
auth_plugin=mysql_native_password
```

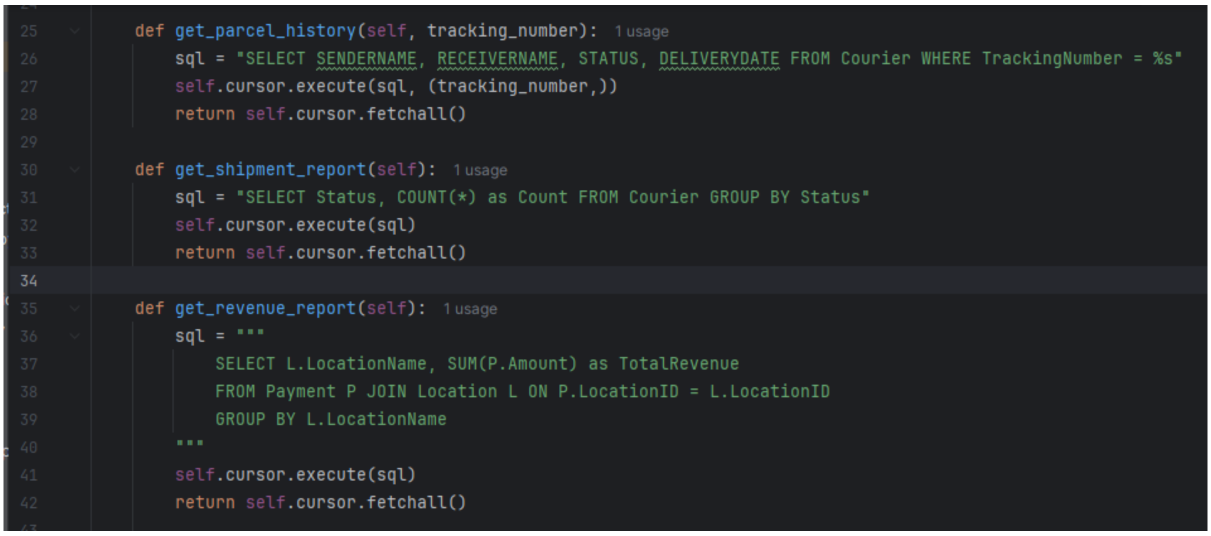

user_service_impl.py
courier_admin_service_impl.py
courier_admin_service_collection_impl.py
db_connection.py

```python
import mysql.connector
from configparser import ConfigParser

class DBConnection:
    connection = None

    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            config = ConfigParser()
            config.read('connectionutil/db.properties')

            DBConnection.connection = mysql.connector.connect(
                host=config.get('DEFAULT', 'host'),
                port=config.get('DEFAULT', 'port'),
                user=config.get('DEFAULT', 'user'),
                password=config.get('DEFAULT', 'password'),
                database=config.get('DEFAULT', 'database'),
                auth_plugin=config.get('DEFAULT', 'auth_plugin')
            )
        return DBConnection.connection
```

2. Create a Service class CourierServiceDb in dao with a static variable named connection of type Connection which can be assigned in the constructor by invoking the method in DBConnection Class.



```python
nectionutil.db_connection import DBConnection

class CourierServiceDb:
    connection = None

    def __init__(self):
        if CourierServiceDb.connection is None:
            CourierServiceDb.connection = DBConnection.get_connection()
        self.cursor = CourierServiceDb.connection.cursor(dictionary=True)

    def insert_courier(self, courier_data):
        sql = """
            INSERT INTO Courier (CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress,
            Weight, Status, TrackingNumber, DeliveryDate, EmployeeID, ServiceID)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s,%s,%s,%s)
        """
        self.cursor.execute(sql, courier_data)
        CourierServiceDb.connection.commit()

    def update_status(self, tracking_number, new_status):
        sql = "UPDATE Courier SET Status = %s WHERE TrackingNumber = %s"
        self.cursor.execute(sql, (new_status, tracking_number))
        CourierServiceDb.connection.commit()
```

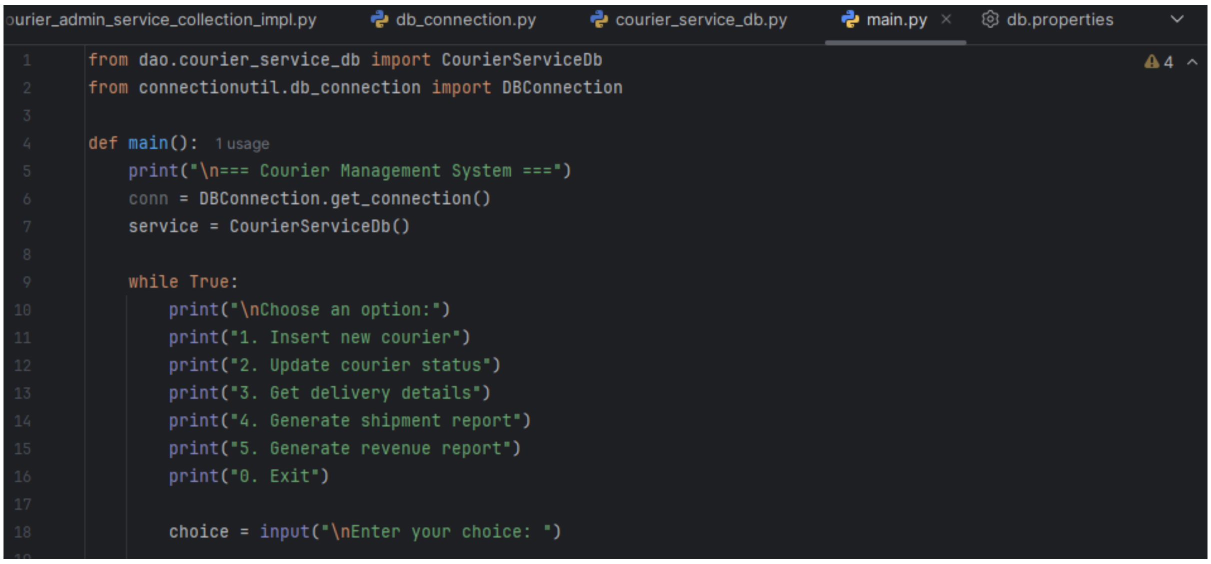


```python
    def get_parcel_history(self, tracking_number):  1 usage
        sql = "SELECT SENDERNAME, RECEIVERNAME, STATUS, DELIVERYDATE FROM Courier WHERE TrackingNumber = %s"
        self.cursor.execute(sql, (tracking_number,))
        return self.cursor.fetchall()

    def get_shipment_report(self):  1 usage
        sql = "SELECT Status, COUNT(*) as Count FROM Courier GROUP BY Status"
        self.cursor.execute(sql)
        return self.cursor.fetchall()

    def get_revenue_report(self):  1 usage
        sql = """
            SELECT L.LocationName, SUM(P.Amount) as TotalRevenue
            FROM Payment P JOIN Location L ON P.LocationID = L.LocationID
            GROUP BY L.LocationName
        """
        self.cursor.execute(sql)
        return self.cursor.fetchall()
```

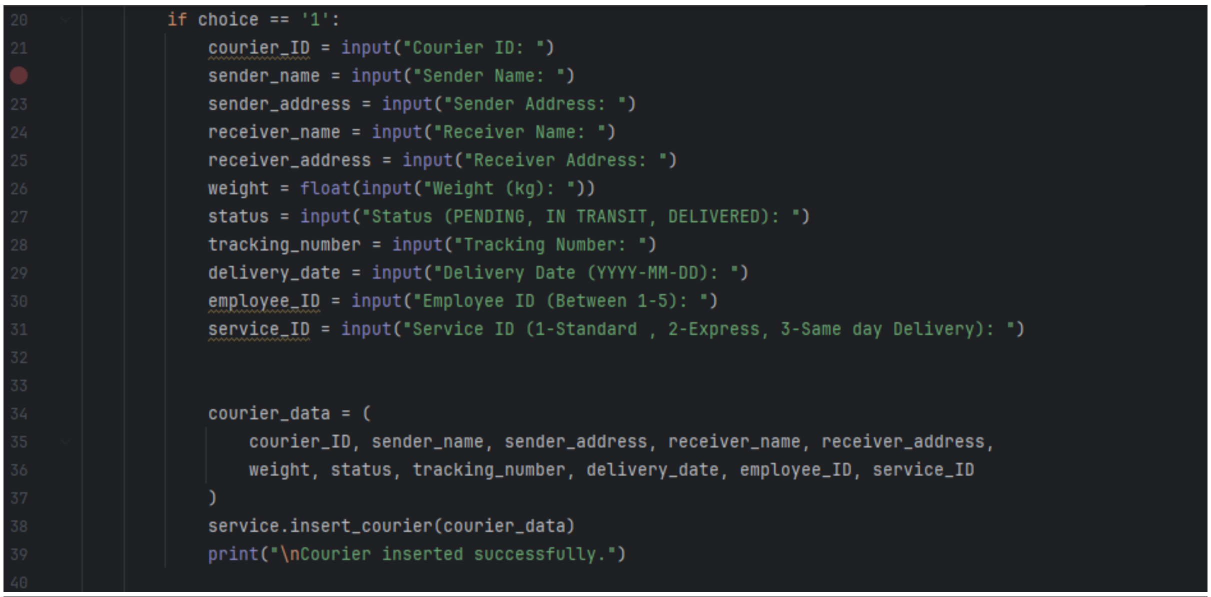
3. Include methods to insert, update, and retrieve data from the database (e.g., inserting a new order, updating courier status).
4. Implement a feature to retrieve and display the delivery history of a specific parcel by querying the database. 1. Generate and display reports using data retrieved from the database (e.g., shipment status report, revenue report).

   The answer for Q3 and Q4 is coded together in main.py

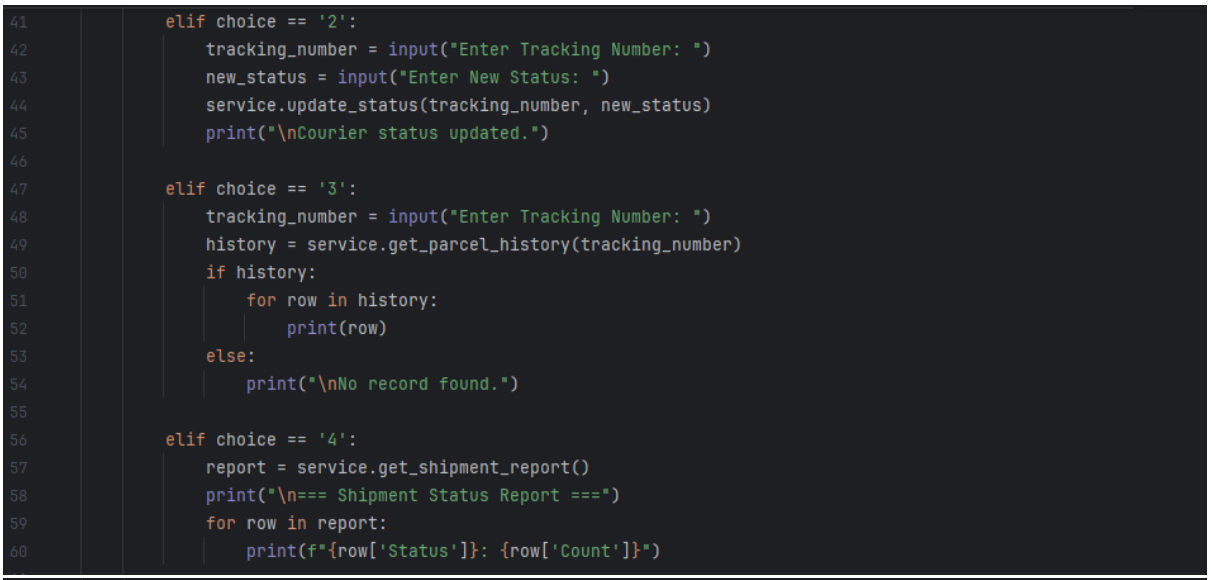
   Code:



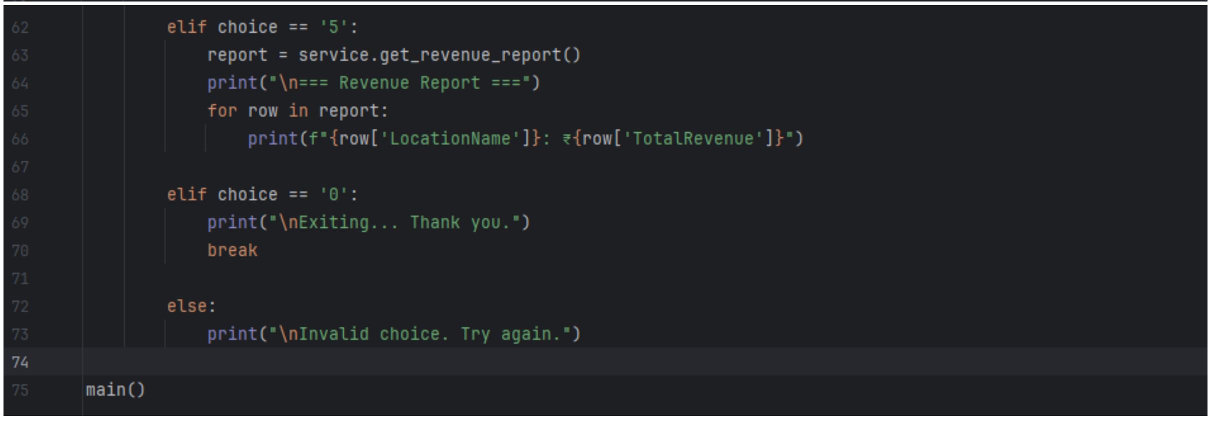
```python
from dao.courier_service_db import CourierServiceDb
from connectionutil.db_connection import DBConnection

def main():  1 usage
    print("\n=== Courier Management System ===")
    conn = DBConnection.get_connection()
    service = CourierServiceDb()

    while True:
        print("\nChoose an option:")
        print("1. Insert new courier")
        print("2. Update courier status")
        print("3. Get delivery details")
        print("4. Generate shipment report")
        print("5. Generate revenue report")
        print("0. Exit")

        choice = input("\nEnter your choice: ")
```



```python
        if choice == '1':
            courier_ID = input("Courier ID: ")
            sender_name = input("Sender Name: ")
            sender_address = input("Sender Address: ")
            receiver_name = input("Receiver Name: ")
            receiver_address = input("Receiver Address: ")
            weight = float(input("Weight (kg): "))
            status = input("Status (PENDING, IN TRANSIT, DELIVERED): ")
            tracking_number = input("Tracking Number: ")
            delivery_date = input("Delivery Date (YYYY-MM-DD): ")
            employee_ID = input("Employee ID (Between 1-5): ")
            service_ID = input("Service ID (1-Standard , 2-Express, 3-Same day Delivery): ")


            courier_data = (
                courier_ID, sender_name, sender_address, receiver_name, receiver_address,
                weight, status, tracking_number, delivery_date, employee_ID, service_ID
            )
            service.insert_courier(courier_data)
            print("\nCourier inserted successfully.")

```

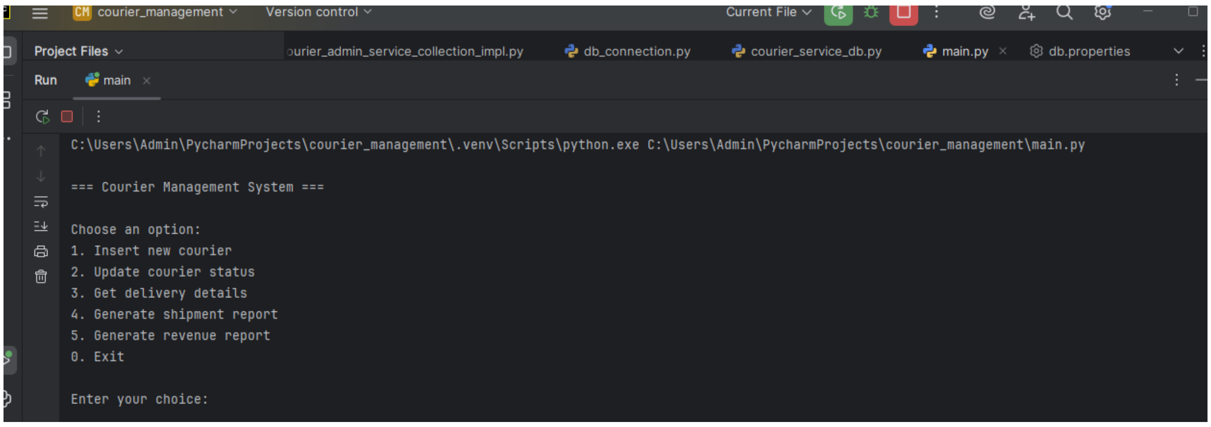


```python
        elif choice == '2':
            tracking_number = input("Enter Tracking Number: ")
            new_status = input("Enter New Status: ")
            service.update_status(tracking_number, new_status)
            print("\nCourier status updated.")

        elif choice == '3':
            tracking_number = input("Enter Tracking Number: ")
            history = service.get_parcel_history(tracking_number)
            if history:
                for row in history:
                    print(row)
            else:
                print("\nNo record found.")

        elif choice == '4':
            report = service.get_shipment_report()
            print("\n=== Shipment Status Report ===")
            for row in report:
                print(f"{row['Status']}: {row['Count']}")
```



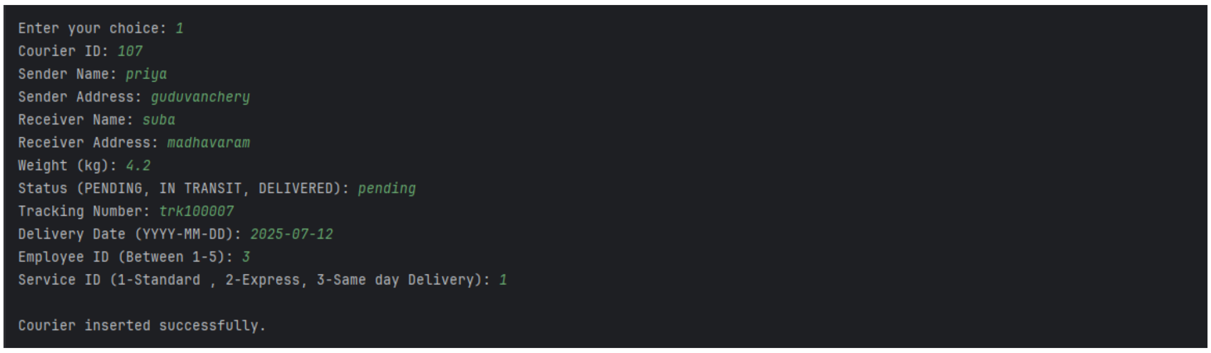
```python
        elif choice == '5':
            report = service.get_revenue_report()
            print("\n=== Revenue Report ===")
            for row in report:
                print(f"{row['LocationName']}: ₹{row['TotalRevenue']}")

        elif choice == '0':
            print("\nExiting... Thank you.")
            break

        else:
            print("\nInvalid choice. Try again.")

main()
```

Output:



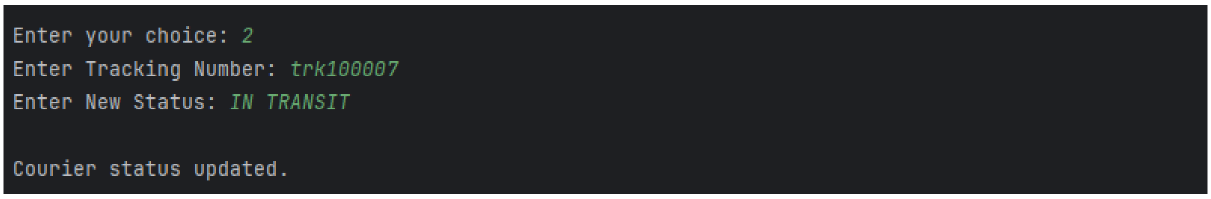
```
C:\Users\Admin\PycharmProjects\courier_management\.venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\courier_management\main.py

=== Courier Management System ===

Choose an option:
1. Insert new courier
2. Update courier status
3. Get delivery details
4. Generate shipment report
5. Generate revenue report
0. Exit

Enter your choice:
```

Inserting new courier:



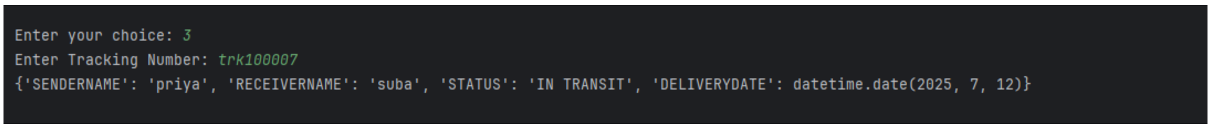
```
Enter your choice: 1
Courier ID: 107
Sender Name: priya
Sender Address: guduvanchery
Receiver Name: suba
Receiver Address: madhavaram
Weight (kg): 4.2
Status (PENDING, IN TRANSIT, DELIVERED): pending
Tracking Number: trk100007
Delivery Date (YYYY-MM-DD): 2025-07-12
Employee ID (Between 1-5): 3
Service ID (1-Standard , 2-Express, 3-Same day Delivery): 1

Courier inserted successfully.
```

Updating courier status:



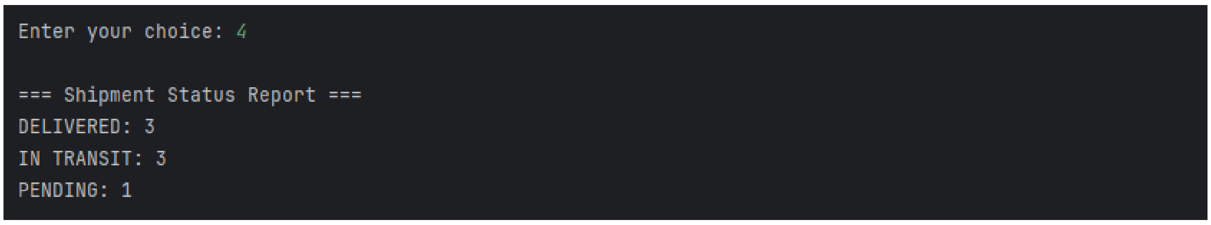
```
Enter your choice: 2
Enter Tracking Number: trk100007
Enter New Status: IN TRANSIT

Courier status updated.
```

Getting delivery details:



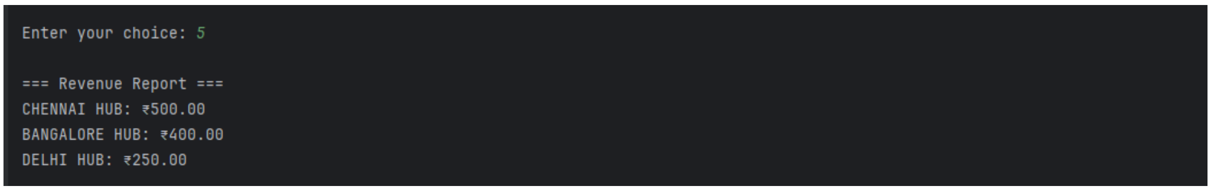
```
Enter your choice: 3
Enter Tracking Number: trk100007
{'SENDERNAME': 'priya', 'RECEIVERNAME': 'suba', 'STATUS': 'IN TRANSIT', 'DELIVERYDATE': datetime.date(2025, 7, 12)}
```

Shipment report:

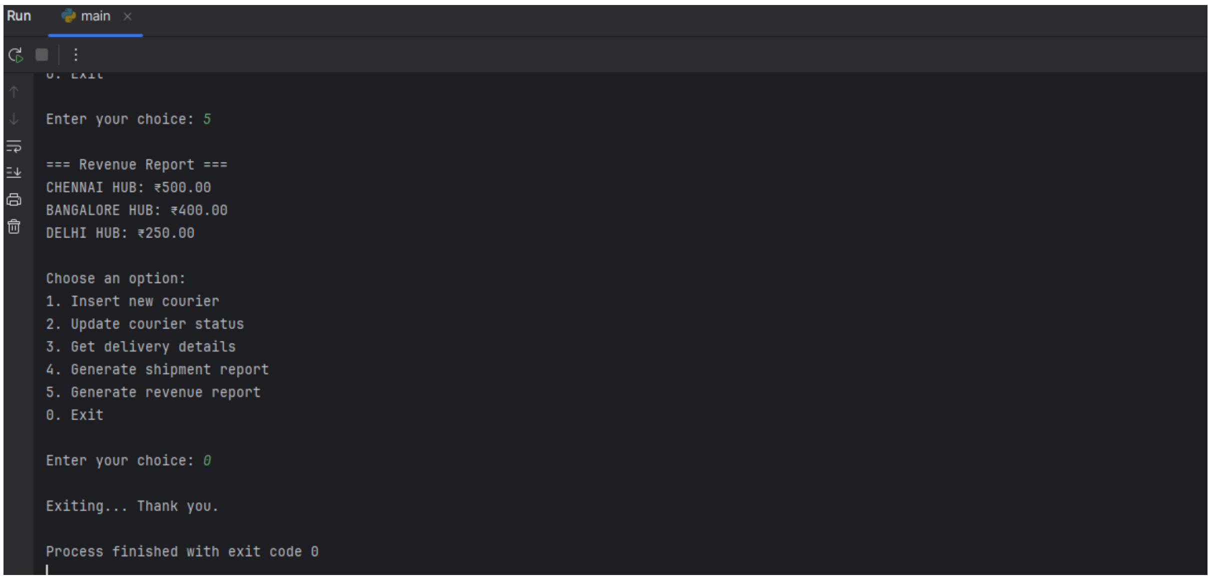


```
Enter your choice: 4

=== Shipment Status Report ===
DELIVERED: 3
IN TRANSIT: 3
PENDING: 1
```

Revenue report:


```
Enter your choice: 5

=== Revenue Report ===
CHENNAI HUB: ₹500.00
BANGALORE HUB: ₹400.00
DELHI HUB: ₹250.00
```

Exit:



```
Enter your choice: 5

=== Revenue Report ===
CHENNAI HUB: ₹500.00
BANGALORE HUB: ₹400.00
DELHI HUB: ₹250.00

Choose an option:
1. Insert new courier
2. Update courier status
3. Get delivery details
4. Generate shipment report
5. Generate revenue report
0. Exit

Enter your choice: 0

Exiting... Thank you.

Process finished with exit code 0
```